aws

开发人员指南

Amazon Rekognition

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Rekognition: 开发人员指南

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 Amazon Rekognition？ 1
关键功能 1
使用案例 2
优势 3
Amazon Rekognition 和 HIPAA 资格 3
您是 Amazon Rekognition 的新用户吗？ 3
Rekognition 的工作原理 5
了解 Rekognition 的分析类型 6
标签 8
自定义标签 9
面部活跃度检测 9
人脸检测和分析 9
人脸搜索 10
人物的轨迹 10
个人防护设备 10
名人 10
文本检测 11
不当或冒犯性内容 11
自定义 11
批量分析 11
了解 Rekognition 的图像和视频操作 12
Amazon Rekognition Image 操作 12
Amazon Rekognition Video 操作 12
基于非存储的操作和基于存储的操作 13
使用 AWS 软件开发工具包或 HTTP 调用 Amazon Rekognition API 操作 13
非存储和存储 API 操作 13
非存储操作 14
基于存储的 API 操作 15
了解模型版本控制 17
入门 18
步骤 1：设置 AWS 账户并创建用户 18
创建 AWS 账户和用户 19
步骤 2：设置 AWS CLI 和 AWS SDKs 21
授予编程式访问权限 23

与... 合作 AWS SDKs 26
第 3 步：开始使用 AWS CLI 和 S AWS DK API 27
格式化示 AWS CLI 例 27
后续步骤 27
步骤 4：开始使用控制台 28
设置控制台权限 28
练习 1：检测对象和场景（控制台） 32
练习 2：分析人脸（控制台） 38
练习 3：比较人脸（控制台） 41
练习 4：查看聚合指标（控制台） 44
使用图像和视频 46
使用图像 46
图像规格 47
分析 Amazon S3 存储桶中的图像 48
使用本地文件系统 64
显示边界框 79
获取图像方向和边界框坐标 91
使用存储视频分析操作 102
分析类型 102
Amazon Rekognition Video API 概述 103
调用 Amazon Rekognition Video 操作 105
配置 Amazon Rekognition Video 111
分析存储视频 (SDK) 114
分析视频（AWS CLI） 143
参考：视频分析结果通知 146
Amazon Rekognition Video 故障排除 147
使用流视频事件 150
Amazon Rekognition Video 流处理器操作概述 150
向新的流处理器添加标签 150
向现有流处理器添加标签 151
列出流处理器中的标签 152
删除流处理器中的标签 152
错误处理 153
错误组成部分 153
错误消息和代码 154
应用程序中的错误处理 158

将 Amazon Rekognition 与 FedRAMP 结合使用 159
传感器、输入图像和视频的最佳实践 163
Amazon Rekognition Image 操作延迟 163
有关面部比较输入图像的建议 164
输入图像进行人脸操作的一般建议 164
在集合中搜索人脸的建议 164
针对摄像机设置（图像和视频）的建议 165
针对摄像机设置（存储视频和流视频）的建议 166
针对摄像机设置（流视频）的建议 167
Face Liveness 的使用建议 167
检测对象和概念 168
为响应对象添加标签 169
边界框 169
置信度得分 170
父级 170
类别 171
别名 171
图像属性 171
模型版本 173
纳入和排除筛选器 173
对结果进行排序和汇总 174
检测图像中的标签 174
DetectLabels 操作请求 185
DetectLabels 响应 186
转变 DetectLabels应对措施 189
检测视频中的标签 193
StartLabelDetection请求 194
GetLabelDetection 操作响应 195
转变回 GetLabelDetection 应 201
检测流视频事件中的标签 209
设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源 210
调用流视频事件的标签检测操作 214
检测自定义标签 220
检测和分析人脸 221
人脸检测和人脸比较概述 222
人脸属性指南 223

检测图像中的人脸 224
DetectFaces 操作请求 236
DetectFaces 操作响应 236
比较图像中的人脸 243
CompareFaces 操作请求 255
CompareFaces 操作响应 255
检测存储视频中的文本 258
GetFaceDetection 操作响应 267
在集合中搜索人脸 273
管理人脸集合、人脸和用户 275
管理集合 275
管理集合中的人脸 276
在集合中管理用户 276
使用相似度阈值关联和匹配人脸 276
使用相似度阈值关联人脸 277
使用相似性阈值匹配人脸 277
常见场景中的人脸索引指南 277
关键或公共安全应用 277
照片共享和社交媒体应用 278
一般使用 278
搜索集合内的人脸和用户 278
涉及公共安全的使用案例 279
使用 Amazon Rekognition 来帮助公共安全 280
创建集合 280
CreateCollection 操作请求 287
CreateCollection 操作响应 287
标记集合 287
向新集合添加标签 287
向现有集合添加标签 288
列出集合中的标签 290
从集合中删除标签 291
列出集合 292
ListCollections 操作请求 298
ListCollections 操作响应 298
描述集合 299
DescribeCollection 操作请求 306

DescribeCollection操作响应 306
删除集合 306
DeleteCollection 操作请求 313
DeleteCollection 操作响应 313
将人脸添加到集合 313
筛选人脸 314
IndexFaces 操作请求 323
IndexFaces 操作响应 324
列出集合中的人脸和相关用户 332
ListFaces 操作请求 338
ListFaces 操作响应 338
从集合中删除人脸 340
DeleteFaces 操作请求 345
DeleteFaces 操作响应 345
创建用户 346
删除用户 349
将人脸与用户关联 352
AssociateFaces 操作响应 355
取消用户人脸关联 356
DisassociateFaces 操作响应 359
列出集合中的用户 360
ListUsers 操作响应 362
搜索人脸（人脸 ID） 363
SearchFaces 操作请求 369
SearchFaces 操作响应 370
搜索人脸（图像） 371
SearchFacesByImage 操作请求 378
SearchFacesByImage 操作响应 378
搜索用户（人脸 ID /用户 ID） 379
SearchUsers 操作请求 383
SearchUsers 操作响应 384
搜索用户（图像） 385
SearchUsersByImage 操作请求 388
SearchUsersByImage 操作响应 389
搜索存储视频中的人脸 390
GetFaceSearch 操作响应 399

在流视频中搜索集合中的人脸 404
设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源 405
在流视频中搜索人脸 408
使用 GStreamer插件进行直播 430
流视频问题排查 432
人物的轨迹 439
GetPersonTracking 操作响应 448
检测个人防护设备 453
个人防护设备的类型 454
口罩 454
手套 454
头罩 454
个人防护设备检测置信度 454
汇总图像中检测到的个人防护设备 455
教程：创建使用 PPE 检测图像的 AWS Lambda 函数 455
了解个人防护设备检测 API 455
提供图片 456
了解回 DetectProtectiveEquipment 应 457
检测图像中的个人防护设备 462
示例：边界框和口罩 474
识别名人 490
名人识别与人脸搜索比较 490
识别图像中的名人 491
正在呼叫 RecognizeCelebrities 491
RecognizeCelebrities 操作请求 500
RecognizeCelebrities 操作响应 501
识别存储视频中的名人 503
GetCelebrityRecognition 操作响应 519
获取名人信息 521
正在呼叫 GetCelebrityInfo 521
GetCelebrityInfo 操作请求 526
GetCelebrityInfo 操作响应 526
审核内容 528
使用图像和视频审核 APIs 529
标签类别 530
内容类型 537

置信度 537
版本控制 537
排序和汇总 537
自定义审核适配器状态 538
测试内容审核版本 7 并转换 API 响应 538
AWS 内容审核 SDK 和使用指南版本 7 539
版本 6.1 到版本 7 的标签映射 539
检测不当图像 543
检测图像中的不当内容 543
..... 543
DetectModerationLabels 操作请求 550
DetectModerationLabels 操作响应 550
检测不当的存储视频 552
GetContentModeration 操作响应 560
使用自定义审核提高准确性 563
创建和使用适配器 563
准备数据集 566
使用 AWS CLI 管理适配器和 SDKs 567
自定义审核适配器教程 573
评估和改进您的适配器 590
清单文件格式 591
训练适配器的最佳实践 596
设置 AutoUpdate权限 597
AWS Rekognition 的健康控制面板通知 599
使用 Amazon A2I 审核不当内容 601
检测文本 606
检测图像中的文本 607
DetectText 操作请求 616
DetectText 操作响应 617
检测存储视频中的文本 622
筛选条件 631
GetTextDetection 响应 632
检测视频分段 638
技术提示 639
黑帧 639
服务抵扣金额 639

彩条 639
画面 639
工作室徽标 639
内容 640
镜头检测 640
关于 Amazon Rekognition Video 分段检测 API 641
使用 Amazon Rekognition 分段 API 641
起始段分析 642
获取分段分析结果 643
示例：检测存储视频中的分段 647
检测人脸活跃度 660
用户端面部活跃度要求 661
架构和序列图 662
先决条件 664
步骤 1：设置 AWS 账户 664
第 2 步：设置 Face Liveness AWS SDKs 665
第 3 步：设置 AWS Amplify 资源 665
检测 Face Liveness 的最佳实践 665
对亚马逊 Rekognition Face Liveness 进行编程 APIs 665
步骤 1: CreateFaceLivenessSession 666
步骤 2: StartFaceLivenessSession 667
步骤 3: GetFaceLivenessSessionResults 667
步骤 4：对结果做出回应 668
直言不讳地活着 APIs 668
配置和自定义您的应用程序 675
配置应用程序 675
自定义您的应用程序 675
Face Liveness 责任共担模式 675
Face Liveness 更新准则 678
版本控制和时间范围 679
版本发布和兼容性矩阵 679
新版本的沟通 680
Face Liveness 常见问题 680
批量分析 683
批量处理图像 683
创建批量分析任务 (CLI) 683

批量分析输出清单 684
内容类型 685
验证预测和训练适配器 686
教程 687
使用 Amazon RDS 和 DynamoDB 存储 Amazon Rekognition 数据 687
先决条件 688
获取 Amazon S3 存储桶中的图像的标签 688
创建 Amazon DynamoDB 表 689
将数据上传到 DynamoDB 691
在 Amazon RDS 中创建 MySQL 数据库 692
将数据上传到 Amazon RDS MySQL 表 694
使用 Amazon Rekognition 和 Lambda 标记 Amazon S3 存储桶中的资产 696
先决条件 697
配置 IAM Lambda 角色 697
创建项目 698
编写代码 701
打包项目 711
部署 Lambda 函数 712
测试 Lambda 方法 713
创建 AWS 视频分析器应用程序 714
先决条件 714
过程 715
创建 Amazon Rekognition Lambda 函数 715
先决条件 717
创建 SNS 主题 717
创建 Lambda 函数 717
配置 Lambda 函数 718
配置 IAM Lambda 角色 719
创建 AWS Toolkit for Eclipse Lambda 项目 720
测试 Lambda 函数 723
使用 Amazon Rekognition 进行身份验证 725
先决条件 725
创建集合 726
新用户注册 727
现有用户登录 734
使用 Amazon Rekognition（REK）检测用于营销应用程序的标签 737

先决条件 738
创建和填充 DynamoDB 表 738
创建具有日志记录和通知功能的 Lambda 函数 740
设置权限和 Lambda 函数 743
使用元数据将图像上传到 Amazon S3 744
来自客户端的轮询 CloudWatch 日志 745
使用 Lambda 和 Python 检测图像中的标签 747
创建 Lambda 函数（控制台） 747
（可选）创建层（控制台） 748
添加 Python 代码（控制台） 750
添加 Python 代码（控制台） 752
代码示例 756
基本功能 760
Hello Amazon Rekognition 761
操作 764
场景 922
构建集合并在其中寻找人脸 922
创建无服务器应用程序来管理照片 935
检测图像中的 PPE 939
检测并显示图像中的元素 939
检测图像中的人脸 955
检测视频中的信息 956
检测图像中的对象 995
检测视频中的人物和对象 998
保存 EXIF 和其他图像信息 1000
API 参考 1001
安全性 1002
身份和访问管理 1002
受众 1003
使用身份进行身份验证 1003
使用策略管理访问 1005
Amazon Rekognition 如何与 IAM 协同工作 1007
AWS 托管式策略 1013
策略最佳实践 1020
使用 控制台 1020
使用基于身份的策略示例 1021

基于资源的策略示例 1023
故障排除 1024
数据保护 1026
数据加密 1027
互联网络流量隐私 1029
将 Amazon Rekognition 与 Amazon VPC 端点结合使用 1030
为 Amazon Rekognition 创建 Amazon VPC 端点 1030
为 Amazon Rekognition 创建 VPC 端点策略 1031
合规性验证 1032
恢复能力 1032
配置和漏洞分析 1033
防止跨服务混淆代理 1033
基础结构安全性 1035
监控 1036
使用亚马逊监控 Rekognition CloudWatch 1036
使用 Rekognition 的 CloudWatch 指标 1036
访问 Rekognition 指标 1038
创建警报 1038
CloudWatchRekognition 的指标 1040
使用记录亚马逊 Rekognition API 调用 AWS CloudTrail 1044
亚马逊 Rekognition 中的信息 CloudTrail 1044
了解 Amazon Rekognition 日志文件条目 1045
准则和配额 1048
支持的区域 1048
设置配额 1048
Amazon Rekognition Image 1048
Amazon Rekognition Image 批量分析 1048
Amazon Rekognition Video 存储视频 1049
Amazon Rekognition Video 流视频 1049
默认限额 1050
计算 TPS 限额更改 1050
TPS 限额的最佳实践 1050
创建更改 TPS 配额的案例 1051
文档历史记录 1053
AWS 词汇表 1062
..... mlxiii

什么是 Amazon Rekognition？

Amazon Rekognition 是一项基于云的图像和视频分析服务，您可以使用该服务向应用程序轻松添加高级计算机视觉功能。该服务由成熟的深度学习技术提供支持，无需机器学习专业知识即可使用。Amazon Rekognition 包含一个 easy-to-use简单的 API，可以快速分析存储在 Amazon S3 中的任何图像或视频文件。

您可以使用 Rekognition 添加检测对象、文本、不安全内容、分析图像/视频以及将人脸与应用程序进行比较的功能。 APIs借助 Amazon Rekognition 的人脸识别 APIs，您可以针对各种用例检测、分析和比较人脸，包括用户验证、编目、人数统计和公共安全。

该服务基于同样由 Amazon 计算机视觉科学家开发的成熟且高度可扩展的深度学习技术，这项技术每天能够分析数十亿图像和视频。Rekognition 定期从新数据中学习，我们会经常向该服务添加新的标签和功能。

有关更多信息，请参阅亚马逊 Rek FAQs ognition。

关键功能

图像分析：

- 对象、场景和概念检测：检测图像中的对象、场景、概念和名人并进行分类。
- 文本检测：检测和识别图像中的各种语言的印刷和手写文本。
- 不安全内容：检测和筛选露骨、不当和暴力的内容和图像。检测精细的不安全内容标签。
- 名人识别：识别图像中成千上万不同类别的名人，例如政治家、运动员、演员和音乐家。
- 人脸分析：检测、分析和比较人脸以及人脸属性，例如性别、年龄和情绪。使用案例可能包括用户验证、编目、人数统计和公共安全。
- 自定义标签：构建自定义分类器以检测特定于使用案例的对象，例如徽标、产品、字符。
- 图像属性：分析图像属性，例如质量、颜色、锐度、对比度。

视频分析：

- 对象、场景和概念检测：检测视频中的对象、场景、概念和名人并进行分类。
- 文本检测：检测和识别视频中的各种语言的印刷和手写文本。

- 人物的轨迹：跟踪已识别的人物在视频帧中的移动。
- 人脸分析：检测、分析和比较流视频或存储视频中的人脸。
- 名人识别：识别存储视频中成千上万不同类别的名人，例如政治家、运动员、演员和音乐家。
- 不安全内容检测：检测视频中露骨、不当和暴力的内容。
- 视频分段：自动识别视频的有用分段，例如黑帧和片尾字幕。
- Face Liveness：在人脸验证期间检测是否有真实用户在场。

使用案例

可搜索媒体库：Rekognition 可检测图像和视频中的标签、对象、概念和场景。您可以根据此视觉内容分析使这些标签可供搜索。对于构建可搜索的图像和视频库很有用。

基于人脸的用户身份验证：通过将图像中的人脸与参考人脸图像进行比较来确认用户身份。对于应用程序中的身份验证很有用。

Face Liveness 检测：Rekognition Face Liveness 是一项完全托管的机器学习（ML）特征，旨在帮助开发人员遏制基于人脸的身份验证中的欺诈行为。该功能可帮助您验证用户是否真实出现在摄像头前，并且不是仿冒用户人脸的不法分子。使用 Rekognition Face Liveness 可以帮助您检测出现在摄像头前的仿冒攻击，例如打印的照片、数码照片/视频或 3D 面具。它还有助于检测绕过摄像头的仿冒攻击，例如直接注入视频采集子系统的预先录制或深度伪造的视频。

人脸搜索：利用 Rekognition，您可以在图像、存储视频和流视频中搜索与存储在称为人脸集合的容器中的人脸匹配的人脸。人脸集合是您拥有和管理的人脸的索引。根据人脸搜索人物时，您需要对人脸进行索引，然后搜索人脸。

不安全内容检测：检测并筛选图像和视频中露骨、不当和暴力的内容。根据业务需求使用标签进行精细筛选。内容审核 API 还返回任何已检测标签（对象和概念）的层级列表以及置信度分数。这些对象/标签指示不安全内容的具体类别，允许精细筛选和管理大量用户生成的内容 (UGC)。您可以使用适配器对内容审核 API 的输出进行自定义，从而提高图像的性能，例如提供作为训练数据的图像。

检测个人防护设备：检测图像中的个人防护设备，以监控各行业的安全合规性。您可以通过检测不当的设备来自动标记不安全状况，并接收有关这些状况的警报，从而提高合规性并改进训练。

名人识别：识别图像和视频中不同类别的名人，例如政治家、运动员、演员和音乐家。您无需提供姓名即可识别名人外表。

文本检测：检测和提取图像中的文本，用于视觉搜索或提取元数据。这适用于不同的字体和样式。检测方向，以处理标牌和横幅上的文字。

自定义标签：识别特定于业务使用案例的自定义对象、概念和场景，例如徽标检测。您可以训练自定义分类器来处理利基或专有对象，与一般分类器相比，可以提高关键对象的准确性。有关更多信息，请参阅《Amazon Rekognition Custom Labels 开发人员指南》中的什么是 Amazon Rekognition Custom Labels？。

优势

将强大的图像和视频分析功能集成到您的应用程序中：无需专业知识即可为应用程序添加准确的图像和视频分析。有了 Amazon Rekognition API，您无需任何机器学习知识即可通过深度学习进行分析。您可以将计算机视觉快速构建到 Web、移动和设备应用程序中。

基于深度学习的图像和视频分析：使用深度学习分析图像和视频，以实现高精度。Amazon Rekognition 可以检测标签、对象、场景、人脸、名人。对结果进行筛选，以包含/排除特定标签。

可扩展的图像分析：分析数百万张图像，以整理海量视觉数据集。可进行扩展，以处理不断增长的图像库和流量。您无需规划容量，只需按实际用量付费。

根据属性分析和筛选图像：按属性（例如质量、颜色和视觉内容）分析和筛选图像，并检测图像的锐度、亮度和对比度。

与其他 AWS 服务集成 ——亚马逊 Rekognition 开箱即用地与 S3 和 Lambda 集成。你可以 APIs 从 Lambda 调用 Amazon Rekognition，在不移动数据的情况下在 Amazon S3 中处理图像。Rekognition 使用 IAM 具有内置的可扩展性和安全性。 AWS

低成本- Pay-as-you-go 定价，没有最低限额或承诺。提供免费套餐，让您轻松入门。采用分级定价，随着使用量的增加，可节省更多成本。与内部解决方案相比，具有成本效益。

轻松自定义：使用适配器为您的使用案例自定义精度。提供示例图像以训练适配器。改进了给定域的对象和标签检测。无需机器学习专业知识即可轻松定制分析。

有关更多信息，请参阅亚马逊 Rek FAQs ognition。

Amazon Rekognition 和 HIPAA 资格

这是一项符合 HIPAA 要求的服务。有关 AWS《1996 年美国健康保险流通与责任法案》(HIPAA) 以及使用 AWS 服务处理、存储和传输受保护的健康信息 (PHI) 的更多信息，请参阅 HIPAA 概述。

您是 Amazon Rekognition 的新用户吗？

如果您是 Amazon Rekognition 的新用户，建议您按顺序阅读以下内容：

1. Amazon Rekognition 的工作原理— 本节介绍各种亚马逊 Rekognition 组件，您可以使用这些组件来创建体验。 end-to-end
2. Amazon Rekognition 入门 – 在本节中，您将设置账户，安装反映您所选语言的软件开发工具包，并测试 Amazon Rekognition API。有关 Amazon Rekognition 支持的编程语言列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。
3. 使用图像 – 本节提供了有关将 Amazon Rekognition 与存储在 Amazon S3 存储桶中的图像以及从本地文件系统加载的图像结合使用的信息。
4. 使用存储视频分析操作 – 本节提供了有关将 Amazon Rekognition 与存储在 Amazon S3 存储桶中的视频结合使用的信息。
5. 使用流视频事件 – 本节提供了有关将 Amazon Rekognition 与流视频结合使用的信息。

Amazon Rekognition 的工作原理

Amazon Rekognition 提供了两个 API 集用于视觉分析：

- Amazon Rekognition Image 用于图像分析
- Amazon Rekognition Video 用于视频分析

图像分析

借助 Amazon Rekognition Image，您的应用程序可以：

- 检测图像中的对象、场景和概念
- 识别名人
- 检测多种语言的文本
- 检测露骨、不当或暴力的内容或图像
- 检测、分析和比较人脸以及人脸属性，例如年龄和情绪
- 检测是否佩戴个人防护设备

使用案例包括增强照片应用程序、对图像进行分类以及对内容进行审核。

视频分析

借助 Amazon Rekognition Video，您的应用程序可以：

- 跨视频帧跟踪人物和对象
- 识别对象
- 识别名人
- 搜索存储视频和流视频，寻找相关的人
- 分析人脸属性，例如年龄和情绪
- 检测露骨、不当或暴力的内容或图像
- 按时间戳和分段对分析结果进行汇总和排序
- 检测流视频中的人物、宠物和包裹

使用案例包括视频分析、对视频进行分类以及筛选不当内容。

主要特征

- 强大的深度学习分析
- 高精度的对象、场景、人脸、文本检测
- 可集成到应用程序中的易于使用的 API
- 根据您的数据调整的可自定义模型
- 可扩展的媒体库分析

Amazon Rekognition 允许您通过训练自定义适配器来提高其某些深度学习模型的准确性。例如，使用 Amazon Rekognition 自定义审核，您可以通过使用图像训练自定义适配器来调整 Amazon Rekognition 的基础图像分析模型。有关更多信息，请参阅使用自定义审核提高准确性。

以下章节将介绍 Amazon Rekognition 提供的分析类型，并概述 Amazon Rekognition Image 和 Amazon Rekognition Video 操作。此外，还涵盖非存储操作与存储操作之间的差别。

要演示 Amazon Rekognition，你可以看到第 3 步：开始使用 AWS CLI 和 AWS SDK API，其中包括在控制台中试用 Rekogniti APIs on。 AWS

主题

- 了解 Rekognition 的分析类型
- 了解 Rekognition 的图像和视频操作
- 了解非存储和存储 API 操作
- 了解模型版本控制

了解 Rekognition 的分析类型

以下是 Amazon Rekognition Image API 和 Amazon Rekognition Video API 可以执行的分析类型。有关的信息 APIs，请参阅了解 Rekognition 的图像和视频操作。

下表列出了与正在使用的媒体类型和使用情况相关的需要使用的操作：

使用场景	媒体类型	运营
审核内容	映像	DetectModerationLabels, StartMediaAnalysisJob,

使用场景	媒体类型	运营
		GetMediaAnalysisJob, ListMediaAnalysisJobs
	存储视频	StartContentModeration, GetContentModeration
身份验证	映像	CreateCollection, CreateUser, IndexFaces, AssociateFaces, SearchFacesByImage, SearchUsersByImage
	存储视频	CreateCollection, IndexFaces, StartFaceSearch, GetFaceSearch
	流视频（检测人脸活跃度）	CreateFaceLivenessSession, StartFaceLivenessSession, GetFaceLivenessSessionResults,
人脸分析	映像	DetectFaces, CompareFaces
	存储视频	StartFaceDetection, GetFaceDetection
	流视频	CreateStreamProcessor, StartStreamProcessor
物体和活动识别	映像	DetectLabels
	存储视频	StartLabelDetection, GetLabelDetection
互联家居	流视频	StartStreamProcessor
媒体分析	存储视频	StartSegmentDetection, GetSegmentDetection

使用场景	媒体类型	运营
工作场所安全	映像	DetectProtectiveEquipment
文本检测	映像	DetectText
	视频	StartTextDetection, GetTextDetection
人物的轨迹	视频	StartPersonTracking, GetPersonTracking
名人识别	映像	RecognizeCelebrities
	视频	StartCelebrityRecognition, GetCelebrityRecognition
自定义标签检测	映像	DetectCustomLabels
	模型训练	参见自定义标签开发人员指南

标签

标签 是指以下任一项目：物体（例如，花、树或桌子）、事件（例如，婚礼、毕业典礼或生日聚会）、概念（例如，风景、傍晚和自然）或活动（例如，跑步或打篮球）。Amazon Rekognition 可以检测图像和视频中的标签。有关更多信息，请参阅 检测对象和概念。

Rekognition 可以检测图像和存储视频中的大量标签。Rekognition 还可以检测流视频中的少量标签。

根据使用案例使用以下操作检测标签：

- 要检测图像中的标签：使用DetectLabels。您可以识别图像属性，例如主图像颜色和图像质量。为此，请DetectLabels使用 wit `IMAGE_PROPERTIES` h 作为输入参数。
- 要检测存储视频中的标签：使用StartLabelDetection。存储视频不支持检测主图像颜色和图像质量。
- 要检测流媒体视频中的标签：使用CreateStreamProcessor。流视频不支持检测主图像颜色和图像质量。

您可以使用纳入和排除筛选选项来指定要为图像和存储视频标签检测返回的标签类型。

自定义标签

Amazon Rekognition Custom Labels 可以通过训练机器学习模型，识别图像中特定于您的业务需求的物体和场景。例如，您可以训练模型来检测徽标或装配线上的工程机器零件。

> **Note**
>
> 有关 Amazon Rekognition Custom Labels 的信息，请参阅 Amazon Rekognition Custom Labels 开发人员指南。

Amazon Rekognition 提供了用于创建、训练、评估和运行机器学习模型的控制台。有关更多信息，请参阅《Amazon Rekognition Custom Labels 开发人员指南》中的 Amazon Rekognition Custom Labels 入门。您还可以使用 Amazon Rekognition Custom Labels API 来训练和运行模型。有关更多信息，请参阅亚马逊 Rekognit ion 开发者指南中的亚马逊 Rekognition 自定义标签 SDK 入门。 CustomLabels

要使用经过训练的模型分析图像，请使用DetectCustomLabels。

面部活跃度检测

Amazon Rekognition Face Liveness 可以帮助您验证正在进行人脸身份验证的用户是否真实出现在摄像头前，并且不是仿冒用户人脸的不法分子。它可以检测出现在摄像头前的仿冒攻击和绕过摄像头的攻击。用户可以通过拍摄简短的自拍视频来完成 Face Liveness 检查，该检查会返回 Liveness 分数。Face Liveness 通过概率计算确定，检查后会返回置信度分数（介于 0-100 之间）。分数越高，对于接受检查的人是真实的就越有信心。

有关 Face Liveness 的更多信息，请参阅检测人脸活跃度。

人脸检测和分析

Amazon Rekognition 可以检测图像和存储视频中的人脸。借助 Amazon Rekognition，您可以获得有关以下内容的信息：

- 在图像或视频中检测到人脸的位置
- 人脸标记，例如眼睛的位置
- 图像中存在人脸遮挡
- 检测到的情绪，例如快乐或悲伤
- 图像中人物的视线方向

您还可以解读人口统计信息，例如性别或年龄。您可以将一个图像中的人脸与另一个图像中检测到的人脸进行比较。还可以存储有关人脸的信息以供以后检索。有关更多信息，请参阅 检测和分析人脸。

要检测图像中的人脸，请使用 DetectFaces。要检测存储视频中的人脸，请使用 StartFaceDetection。

人脸搜索

Amazon Rekognition 可以搜索人脸。人脸信息将索引到称为集合的容器中。集合中的人脸信息随后可与在图像、存储视频和流视频中检测到的人脸进行匹配。有关更多信息，请参阅 在集合中搜索人脸。

要搜索图像中的已知人脸，请使用 DetectFaces。要搜索存储视频中的已知人脸，请使用 StartFaceDetection。要搜索流视频中的已知人脸，请使用 CreateStreamProcessor。

人物的轨迹

Amazon Rekognition 可以跟踪在存储视频中检测到的人物的轨迹。Amazon Rekognition Video 提供了在视频中检测到的人物的跟踪轨迹、人脸详细信息以及帧内位置信息。有关更多信息，请参阅 人物的轨迹。

要检测所存储视频中的人员，请使用 StartPersonTracking。

个人防护设备

Amazon Rekognition 可以检测图像中检测到的人员佩戴的个人防护设备 (PPE)。Amazon Rekognition 可以检测口罩、手套和头罩。Amazon Rekognition 可以预测个人防护设备能否覆盖相应的身体部位。您还可以获得用于检测到的人员和个人防护装备物品的边界框。有关更多信息，请参阅 检测个人防护设备。

要检测图像中的 PPE，请使用DetectProtectiveEquipment。

名人

Amazon Rekognition 可以识别图像和存储视频中的数千位名人。您可以获取有关名人的人脸在图像上的位置、人脸标记以及名人人脸的姿势的信息。您可以获取名人出现在存储视频中时的跟踪信息。您还可以获得有关知名名人的更多信息，例如所表达的情感和性别的表现。有关更多信息，请参阅 识别名人。

要识别图像中的名人，请使用 RecognizeCelebrities。要识别存储视频中的名人，请使用 StartCelebrityRecognition。

文本检测

Amazon Rekognition 图像文本识别可以检测图像中的文本并将其转换为机器可读的文本。有关更多信息，请参阅 检测文本。

要检测图像中的文本，请使用 DetectText。

不当或冒犯性内容

Amazon Rekognition 可以分析图像和存储视频中是否有成人和暴力内容。有关更多信息，请参阅 审核内容。

要检测不安全的图像，请使用 DetectModerationLabels。要检测不安全的存储视频，请使用 StartContentModeration。

自定义

Rekognition APIs 提供的某些图像分析允许您通过创建根据自己的数据训练的自定义适配器来提高深度学习模型的准确性。适配器是可插入 Rekognition 预先训练好的深度学习模型的组件，可利用基于图像的领域知识提高模型的准确性。您可以通过提供样本图像并对其进行注释来训练适配器以满足您的需求。

创建适配器后，系统会为您提供一个 AdapterId。您可以将其提供 AdapterId 给操作，以指定要使用已创建的适配器。例如，您可以向 DetectModerationLabelsAPI 提供用于同步图像分析。 AdapterId 在请求中提供后 AdapterId ，Rekognition 会自动使用它来增强对图像的预测。这使您可以利用 Rekognition 的功能，同时对其进行自定义以满足您的需求。

您还可以选择使用 StartMediaAnalysisJobAPI 批量获取图像的预测结果。有关更多信息，请参阅批量分析。

您可以通过将图像上传到 Rekognition 控制台并对这些图像进行分析来评测 Rekognition 操作的准确性。Rekognition 将使用所选功能为您的图像添加注释，然后您可以查看预测，使用经过验证的预测来确定哪些标签会从创建适配器中受益。

目前，您可以将适配器与 DetectModerationLabels. 有关创建和使用适配器的更多信息，请参阅使用自定义审核提高准确性。

批量分析

Rekognition Bulk Analysis 允许您在执行操作时使用清单文件异步处理大量图像。StartMediaAnalysisJob有关更多信息，请参阅批量分析。

了解 Rekognition 的图像和视频操作

Amazon Rekognition 提供两个主要 API 集用于进行图像和视频分析：

- Amazon Rekognition Image：此 API 旨在分析图像。
- Amazon Rekognition Video：此 API 侧重于分析存储视频和流视频。

两者 APIs 都可以检测各种实体，例如人脸和物体。要全面了解支持的比较和检测类型，请参阅[了解 Rekognition 的分析类型]中的章节。

Amazon Rekognition Image 操作

Amazon Rekognition Image 操作是同步的。输入和响应采用 JSON 格式。Amazon Rekognition Image 操作将分析采用 .jpg 或 .png 图像格式的输入图像。传递到 Amazon Rekognition Image 操作的图像可存储在 Amazon S3 存储桶中。如果您未使用 AWS CLI，则还可以将 Base64 编码图像字节直接传递到 Amazon Rekognition 操作。有关更多信息，请参阅[使用图像]。

Amazon Rekognition Video 操作

Amazon Rekognition Video API 便于分析存储在 Amazon S3 存储桶中的视频，或通过 Amazon Kinesis Video Streams 流式传输的视频。

对于存储视频操作，请注意以下几点：

- 操作是异步的。
- 分析必须通过 “开始” 操作启动（StartFaceDetection例如，在存储的视频中进行人脸检测）。
- 分析的完成状态将发布到 Amazon SNS 主题。
- 要检索分析结果，请使用相应的 “获取” 操作（例如 GetFaceDetection）。
- 有关更多信息，请参阅[使用存储视频分析]。

对于流视频分析：

- 功能包括 Rekognition Video 集合中的人脸搜索和标签（对象或概念）检测。
- 标签分析结果将作为 Amazon SNS 和 Amazon S3 通知发送。
- 人脸搜索结果将输出到 Kinesis 数据流。
- 流媒体视频分析的管理是通过 Amazon Rekognition Video 流处理器完成的（例如，使用创建处理器）。CreateStreamProcessor

- 有关更多信息，请参阅使用流视频事件。

每个视频分析操作都会返回有关正在分析的视频的元数据，以及任务 ID 和任务标签。标签检测和内容审核等视频操作允许您按时间戳或标签名称进行排序，并按时间戳或分段对结果进行汇总。

基于非存储的操作和基于存储的操作

Amazon Rekognition 操作分为以下类别。

- 非存储 API 操作 – 在这些操作中，Amazon Rekognition 不保存任何信息。您提供输入图像和视频，操作执行分析，然后返回结果，但 Amazon Rekognition 不会保存任何信息。有关更多信息，请参阅非存储操作。
- 基于存储的 API 操作 – Amazon Rekognition 服务器可将检测到的人脸信息存储在称为集合的容器中。Amazon Rekognition 提供其他 API 操作，这些操作可用于搜索保存的人脸信息中是否存在匹配的人脸。有关更多信息，请参阅 基于存储的 API 操作。

使用 AWS 软件开发工具包或 HTTP 调用 Amazon Rekognition API 操作

您可以使用 AWS 软件开发工具包调用 Amazon Rekognition API 操作，也可以直接使用 HTTP 调用这些操作。您应总是使用 AWS 软件开发工具包，除非您有充分的理由不这样做。本节中的 Java 示例使用 AWS 软件开发工具包。虽然未提供 Java 项目文件，但您可使用 AWS Toolkit for Eclipse 通过 Java 开发 AWS 应用程序。

本节中的 .NET 示例使用 适用于 .NET 的 AWS SDK。您可以使用 AWS Toolkit for Visual Studio 通过 .NET 开发 AWS 应用程序。它包括有用模板和 AWS 各区服务浏览器，用于部署应用程序和管理服务。

本指南中的 API 参考介绍了使用 HTTP 调用 Amazon Rekognition 操作。有关 Java 参考信息，请参阅适用于 Java 的 AWS SDK。

您可使用的 Amazon Rekognition 服务端点记录在 AWS 区域和端点上。

在使用 HTTP 调用 Amazon Rekognition 时，可使用 POST HTTP 操作。

了解非存储和存储 API 操作

Amazon Rekognition 提供了两种类型的 API 操作。即非存储操作和存储操作。对于前者，Amazon Rekognition 不存储任何信息；对于后者，Amazon Rekognition 存储特定人脸信息。

非存储操作

Amazon Rekognition 针对图像提供了以下非存储 API 操作：

- DetectLabels
- DetectFaces
- CompareFaces
- DetectModerationLabels
- DetectProtectiveEquipment
- RecognizeCelebrities
- DetectText
- GetCelebrityInfo

Amazon Rekognition 针对视频提供了以下非存储 API 操作：

- StartLabelDetection
- StartFaceDetection
- StartPersonTracking
- StartCelebrityRecognition
- StartContentModeration

这些操作称作非存储 API 操作，因为当您进行操作调用时，Amazon Rekognition 不会保留发现的有关输入图像的任何信息。与所有其他 Amazon Rekognition API 操作类似，非存储 API 操作不保留任何输入图像字节。

以下示例方案演示可将非存储 API 操作集成到应用程序中的情况。这些方案假定您有一个本地图像存储库。

Example 1：一个在包含特定标签的本地存储库中查找图像的应用程序

首先，您使用 Amazon Rekognition `DetectLabels` 操作在存储库中的每个图像中检测标签（物体和概念）并建立客户端索引，如下所示：

```
Label        ImageID

tree          image-1
```

```
flower        image-1
mountain      image-1
tulip         image-2
flower        image-2
apple         image-3
```

然后，您的应用程序可以搜索此索引以在本地存储库中查找包含特定标签的图像。例如，显示包含树的图像。

Amazon Rekognition 检测到的每个标签均有一个关联的置信度值。该值指示包含该标签的输入图像的置信度级别。您可以使用此置信度值选择性地对标签执行其他客户端筛选，具体取决于有关检测中的置信度级别的应用程序要求。例如，如果您需要准确的标签，您可以筛选和选择仅具有更高置信度（例如 95% 或更高）的标签。如果您的应用程序不需要更高的置信度值，您可以选择筛选具有较小置信度值（接近 50%）的标签。

Example 2：一个用于显示增强的人脸图像的应用程序

首先，您可以使用 Amazon Rekognition `DetectFaces` 操作在本地存储库中的每个图像中检测人脸，然后建立客户端索引。对于每个人脸，此操作将返回元数据，其中包含边界框、人脸标记（例如，嘴巴和耳朵的位置）和人脸属性（例如，性别）。您可以在客户端本地索引中存储此元数据，如下所示：

```
ImageID     FaceID     FaceMetaData

image-1     face-1     <boundingbox>, etc.
image-1     face-2     <boundingbox>, etc.
image-1     face-3     <boundingbox>, etc.
...
```

在此索引中，主键是 `ImageID` 和 `FaceID` 的组合。

然后，您可以使用索引中的信息，在应用程序从本地存储库显示图像时增强图像。例如，您可以在人脸周围添加边界框或突出显示人脸特征。

基于存储的 API 操作

Amazon Rekognition Image IndexFaces支持该操作，您可以使用该操作来检测图像中的人脸并保留有关在亚马逊 Rekognition 集合中检测到的面部特征的信息。这是基于存储的 API 操作的示例，因为服务会将信息保留在服务器上。

Amazon Rekognition Image 提供了以下存储 API 操作：

- IndexFaces
- ListFaces
- SearchFacesByImage
- SearchFaces
- DeleteFaces
- DescribeCollection
- DeleteCollection
- ListCollections
- CreateCollection

Amazon Rekognition Video 提供了以下存储 API 操作：

- StartFaceSearch
- CreateStreamProcessor

要存储人脸信息，您必须先在账户的一个 AWS 区域中创建一个人脸集合。在调用 `IndexFaces` 操作时指定此人脸集合。在创建人脸集合并存储所有人脸的人脸特征信息后，您可以在集合中搜索匹配的人脸。例如，您可以检测图像中的最大人脸并通过调用 `searchFacesByImage.` 来搜索集合中匹配的人脸。

由 `IndexFaces` 存储在集合中的人脸信息可供 Amazon Rekognition Video 操作访问。例如，您可以通过调用 StartFaceSearch 在视频中搜索其人脸与现有集合中的人脸匹配的人员。

有关创建和管理集合的信息，请参阅在集合中搜索人脸。

> **Note**
>
> 集合存储人脸向量，这些向量是人脸的数学表示。集合不存储人脸图像。

Example 1：用于对进出大楼进行身份验证的应用程序

首先，您应使用 `IndexFaces` 操作来创建一个人脸集合来存储扫描到的徽章图像，此操作将提取人脸并将其存储为可搜索的图像向量。然后，当员工进入大楼时，将捕获员工的人脸图像，并将此图像发送

到 SearchFacesByImage 操作。如果人脸匹配得到了足够高的相似度得分（例如 99%），则可对员工进行身份验证。

了解模型版本控制

Amazon Rekognition 使用深度学习模型来执行人脸检测和搜索集合中的人脸。它根据客户反馈和深度学习研究的进展来提高其模型的准确性。这些改进随模型更新提供。例如，在此模型的 1.0 版本中，IndexFaces 可以为一个图像中的 15 个最大的人脸编制索引。此模型的较新版本支持 IndexFaces 为一个图像中的 100 个最大的人脸编制索引。

当您创建新集合时，它会与此模型的最新版本关联。为了提高准确性，此模型偶尔会更新。

当发布此模型的新版本时，会发生以下情况：

- 您创建的新集合与最新模型关联。您使用 IndexFaces 添加到新集合的人脸通过最新模型来检测。
- 您的现有集合继续使用创建它们时所使用的模型版本。这些集合中存储的人脸向量不会自动更新到此模型的最新版本。
- 已添加到现有集合的新人脸可通过已与该集合关联的模型来检测。

此模型的不同版本互相不兼容。具体而言，如果将一个图像索引到使用此模型的不同版本的多个集合，则检测到的同一人脸的人脸标识符是不同的。如果将一个图像索引到与同一模型关联的多个集合，则人脸标识符是相同的。

如果您的集合管理不负责此模型的更新，您的应用程序可能会遇到兼容性问题。您可以通过使用由集合操作（例如 CreateCollection）的响应中返回的 FaceModelVersion 字段来确定集合使用的模型版本。您可以通过调用 DescribeCollection 获取现有集合的模型版本。有关更多信息，请参阅 描述集合。

集合中的现有人脸向量无法更新为此模型的更高版本。由于 Amazon Rekognition 不会存储源图像字节，因此它不会通过使用此模型的更高版本来自动为图像重建索引。

要对存储在现有集合中的人脸使用最新模型，请创建一个新集合 (CreateCollection) 并将源图像重新索引到该新集合 (Indexfaces) 中。您需要更新由您的应用程序存储的任何人脸标识符，因为新集合中的人脸标识符与旧集合中的人脸标识符是不同的。如果您不再需要旧集合，则可以使用 DeleteCollection 将其删除。

无状态操作（如 DetectFaces）使用最新版本的模型。

Amazon Rekognition 入门

此部分提供了一些主题，可帮助您开始使用 Amazon Rekognition。如果您是首次使用 Amazon Rekognition，建议您先查看Amazon Rekognition 的工作原理中提供的概念和术语。

在开始使用 Rekognition 之前，您需要创建 AWS 账户并获取 AWS 账户 ID。您还需要创建一个用户，这样 Amazon Rekognition 系统就可以确定您是否拥有访问其资源所需的权限。

创建账户后，您需要安装和配置 AWS CLI 和 AWS SDKs。 AWS CLI 允许你通过命令行与亚马逊 Rekognition 和其他服务进行交互， AWS SDKs 而允许你使用 Java 和 Python 等编程语言与亚马逊 Rekognition 进行交互。

设置 AWS CLI 和后 AWS SDKs，您可以查看一些如何同时使用两者的示例。您还可以查看一些有关如何使用控制台与 Amazon Rekognition 互动的示例。

Amazon Rekognition 控制台允许您使用批量分析和自定义审核工作流。除了这些功能外，您还可以使用该控制台演示许多不同的 Amazon Rekognition 特征。请注意，需要使用该 API，才能使用大多数 Amazon Rekognition 特征。

主题

- 步骤 1：设置 AWS 账户并创建用户
- 步骤 2：设置 AWS CLI 和 AWS SDKs
- 第 3 步：开始使用 AWS CLI 和 S AWS DK API
- 步骤 4：开始使用 Amazon Rekognition 控制台

步骤 1：设置 AWS 账户并创建用户

首次使用 Amazon Rekognition 前，您必须完成以下任务：

1. 注册一个 AWS 账户。
2. 创建用户。

开发人员指南的这一部分解释了创建 AWS 账户和用户的原因和方式。

主题

- 创建 AWS 账户和用户

创建 AWS 账户和用户

AWS 账户

在注册 Amazon Web Services (AWS) 时，您的 AWS 账户会自动注册 AWS 中的所有服务，包括 Amazon Rekognition。您只需为使用的服务付费。

使用 Amazon Rekognition 时，您仅需为实际使用的资源付费。

如果你是新 AWS 客户，你可以免费开始使用亚马逊 Rekognition。有关更多信息，请参阅 AWS 免费使用套餐。

有关账户创建说明，请参阅即将发布的注册获取 AWS 账户部分。

如果您已经有一个 AWS 帐户，请跳过帐户设置并创建管理员用户。

Users

AWS 中的服务（如 Amazon Rekognition）要求您在访问时提供凭证。这样，服务才能确定您是否有权访问该服务所拥有的资源。

您可以为 AWS 账户创建访问密钥以访问 AWS CLI 或在使用控制台 APIs 时需要输入密码。但是，我们不建议使用您的 AWS 账户根用户的凭证访问 AWS。相反，我们建议您使用 AWS Identity and Access Management (IAM) 来创建管理用户。

您随后可以使用特殊 URL 和管理用户的凭证访问 AWS。

如果您已注册 AWS 但尚未为自己创建一个用户，则可以使用 IAM 控制台自行创建。有关如何创建管理用户的说明，请参阅即将发布的创建具有管理访问权限的用户部分。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

步骤 2：设置 AWS CLI 和 AWS SDKs

主题

- 授予编程式访问权限
- 将 Rekognition 与 SDK 配合使用 AWS

以下步骤向您展示如何安装 AWS Command Line Interface (AWS CLI) 以及本文档中的示例所 AWS SDKs 使用的方法。有多种不同的方法可以对 AWS SDK 调用进行身份验证。本指南中的示例假设您在调用 AWS CLI 命令和 AWS SDK API 操作时使用默认凭据配置文件。

有关可用 AWS 区域的列表，请参阅中的区域和终端节点Amazon Web Services 一般参考。

按照以下步骤下载和配置 AWS SDKs。

要设置 AWS CLI 和 AWS SDKs

1. 下载AWS CLI并安装您要使用的和。 AWS SDKs 本指南提供了 Java AWS CLI、Python、Ruby、Node.js、PHP、.NET 和的示例 JavaScript。有关安装的信息 AWS SDKs，请参阅 Amazon Web Services 工具。
2. 为您在创建 AWS 账户和用户中创建的用户创建一个访问密钥。

   a. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。

   b. 在导航窗格中，选择 Users（用户）。

c. 选择您在创建 AWS 账户和用户中创建的用户的名称。

d. 选择安全凭证选项卡。

e. 选择创建访问密钥。然后，选择下载 .csv 文件，将访问密钥 ID 和秘密访问密钥保存至计算机上的 CSV 文件中。将文件存储在安全位置。关闭此对话框后，您将无法再次访问该秘密访问密钥。下载 CSV 文件之后选择 Close。

3. 如果您已经安装了 AWS CLI，则可以通过在命令提示符下输入 AWS SDKs `aws configure`来配置大多数人的凭据和区域。否则，请使用以下说明。

4. 在您的计算机上，导航至主目录，并创建 `.aws` 目录。在基于 Unix 的系统（例如 Linux 或 macOS）上，它在以下位置：

```
~/.aws
```

在 Windows 上，它在以下位置：

```
%HOMEPATH%\.aws
```

5. 在 `.aws` 目录中，创建名为 `credentials` 的新文件。

6. 打开您在步骤 2 中创建的凭证 CSV 文件，并使用以下格式将此文件的内容复制到`credentials`文件：

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

用您的访问密钥 ID 和秘密访问密钥替换 your_access_key_id 和 your_secret_access_key。

7. 保存 `Credentials` 文件并删除此 CSV 文件。

8. 在 `.aws` 目录中，创建名为 `config` 的新文件。

9. 打开 `config` 文件并按以下格式输入您的区域。

```
[default]
region = your_aws_region
```

用所需的 AWS 区域（例如，`us-west-2`）替换 your_aws_region。

> **Note**
>
> 如果您未 选择区域，则默认情况下将使用us-east-1。

10. 保存 `config` 文件。

授予编程式访问权限

您可以在本地计算机或其他 AWS 环境（例如 Amazon Elastic Compute Cloud 实例）上运行本指南中的 AWS CLI 和代码示例。要运行示例，您需要授予对示例使用的 AWS SDK 操作的访问权限。

主题

- 在本地计算机上运行代码
- 在 AWS 环境中运行代码

在本地计算机上运行代码

要在本地计算机上运行代码，我们建议您使用短期凭证向用户授予对 AWS SDK 操作的访问权限。有关在本地计算机上运行 AWS CLI 和代码示例的具体信息，请参阅在本地计算机上使用配置文件。

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的 “配置 AWS CLI 要使用”。

哪个用户需要编程式访问权限？	目的	方式
		• 有关工具和 AWS SDKs AWS APIs，请参阅《工具参考指南》中的 IAM 身份中心身份验证AWSSDKs 和工具参考指南。
IAM	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDKs 和工具，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的使用长期凭证进行身份验证。 • 有关信息 AWS APIs，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

在本地计算机上使用配置文件

您可以使用您在中创建 AWS CLI 的短期证书运行本指南中的和代码示例在本地计算机上运行代码。为了获取凭证和其他设置信息，这些示例使用名为 `profile-name` 的配置文件。例如：

```
session = boto3.Session(profile_name="profile-name")
rekognition_client = session.client("rekognition")
```

个人资料所代表的用户必须有权调用 Rekognition SDK 操作以及示例所需的 AWS 其他 SDK 操作。

要创建适用于 AWS CLI 和代码示例的配置文件，请选择以下选项之一。确保您创建的配置文件的名称为 `profile-name`。

- 由 IAM 管理的用户 — 按照切换到 IAM 角色 (AWS CLI) 部分的说明进行操作。
- 员工身份（由管理的用户 AWS IAM Identity Center）— 按照配置要使用的 AWS CLI 中的说明进行操作 AWS IAM Identity Center。对于这些代码示例，我们建议使用集成式开发环境 (IDE)，该环境支持 AWS Toolkit，可通过 IAM Identity Center 实现身份验证。有关 Java 示例，请参阅使用 Java 开始构建。有关 Python 示例，请参阅使用 Python 开始构建。有关更多信息，请参阅 IAM Identity Center 凭证。

> **Note**
>
> 您可以使用代码获取短期凭证。有关更多信息，请参阅切换到 IAM 角色 (AWS API)。对于 IAM Identity Center，请按照获取用于 CLI 访问的 IAM 角色凭证部分的说明操作，获取角色的短期凭证。

在 AWS 环境中运行代码

您不应使用用户凭据在 AWS 环境中签署 AWS SDK 调用，例如在 AWS Lambda 函数中运行的生产代码。相反，您应该配置一个角色来定义代码所需的权限。然后，将该角色附加到运行代码的环境。关于如何附加角色和提供可用的临时凭证，取决于运行代码的环境：

- AWS Lambda 函数 — 使用 Lambda 在您的函数担任 Lambda 函数的执行角色时自动提供给您的函数的临时证书。这些凭证在 Lambda 环境变量中可用。您不需要指定配置文件。有关更多信息，请参阅 Lambda 执行角色。
- 亚马逊 EC2 -使用亚马逊 EC2 实例元数据终端节点凭证提供程序。提供商使用您附加到亚马逊实例的亚马逊 EC2 实例配置文件自动为您生成和刷新证书。 EC2 有关更多信息，请参阅使用 IAM 角色向在 Amazon EC2 实例上运行的应用程序授予权限
- Amazon Elastic Container Service — 使用 Container 凭证提供程序。Amazon ECS 会向元数据端点发送和刷新凭证。您指定的任务 IAM 角色会提供一项策略，用于管理您的应用程序所使用的凭证。有关更多信息，请参阅与 AWS 服务交互。

有关凭证提供程序的更多信息，请参阅标准化凭证提供程序。

将 Rekognition 与 SDK 配合使用 AWS

AWS 软件开发套件 (SDKs) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
适用于 C++ 的 AWS SDK	适用于 C++ 的 AWS SDK 代码示例
AWS CLI	AWS CLI 代码示例
适用于 Go 的 AWS SDK	适用于 Go 的 AWS SDK 代码示例
适用于 Java 的 AWS SDK	适用于 Java 的 AWS SDK 代码示例
适用于 JavaScript 的 AWS SDK	适用于 JavaScript 的 AWS SDK 代码示例
适用于 Kotlin 的 AWS SDK	适用于 Kotlin 的 AWS SDK 代码示例
适用于 .NET 的 AWS SDK	适用于 .NET 的 AWS SDK 代码示例
适用于 PHP 的 AWS SDK	适用于 PHP 的 AWS SDK 代码示例
AWS Tools for PowerShell	AWS Tools for PowerShell 代码示例
适用于 Python (Boto3) 的 AWS SDK	适用于 Python (Boto3) 的 AWS SDK 代码示例
适用于 Ruby 的 AWS SDK	适用于 Ruby 的 AWS SDK 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

有关特定于 Rekognition 的示例，请参阅使用 Amazon Rekognition 的代码示例 AWS SDKs。

> **示例可用性**
>
> 找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

第 3 步：开始使用 AWS CLI 和 S AWS DK API

设置好要使用的 AWS CLI 和之后 AWS SDKs ，您可以构建使用 Amazon Rekognition 的应用程序。尽管只有几个特定的 Amazon Rekognition 特征具有控制台工作流，但与 Amazon Rekognition 的大多数交互都是通过 API 操作进行的。

以下主题向你展示了如何通过或开始使用亚马逊 Rekognition Image 和亚马逊 Rekognition Video。AWS CLI AWS SDKs

- 使用图像：介绍使用 Amazon Rekognition Image 分析图像的过程。
- 使用存储视频分析操作：介绍使用 Amazon Rekognition Video 分析存储的非流视频的过程。
- 使用流视频事件：介绍使用 Amazon Rekognition Video 分析流视频的过程。

上面列出的章节包含使用 AWS CLI的示例。如果您打算使用 AWS CLI，请参阅以下部分，了解有关如何格式化 API 调用的信息。

格式化示 AWS CLI 例

本指南中的 AWS CLI 示例是针对 Linux 操作系统进行格式化的。要将示例用于 Microsoft Windows，您需要更改 `--image` 参数的 JSON 格式，并将换行符从反斜杠 (\) 更改为插字号 (^)。有关 JSON 格式的更多信息，请参阅为 AWS 命令行界面指定参数值。

以下是针对 Microsoft Windows 进行格式化的示例 AWS CLI 命令（请注意，这些命令不会按原样运行，它们只是格式化示例）：

```
aws rekognition detect-labels ^
  --image "{\"S3Object\":{\"Bucket\":\"photo-collection\",\"Name\":\"photo.jpg\"}}" ^
  --region region-name
```

您也可以提供适用于 Microsoft Windows 和 Linux 的速记版 JSON。

```
aws rekognition detect-labels --image "S3Object={Bucket=photo-
collection,Name=photo.jpg}" --region region-name
```

有关更多信息，请参阅将快速输入语法用于 AWS 命令行界面。

后续步骤

步骤 4：开始使用 Amazon Rekognition 控制台

步骤 4：开始使用 Amazon Rekognition 控制台

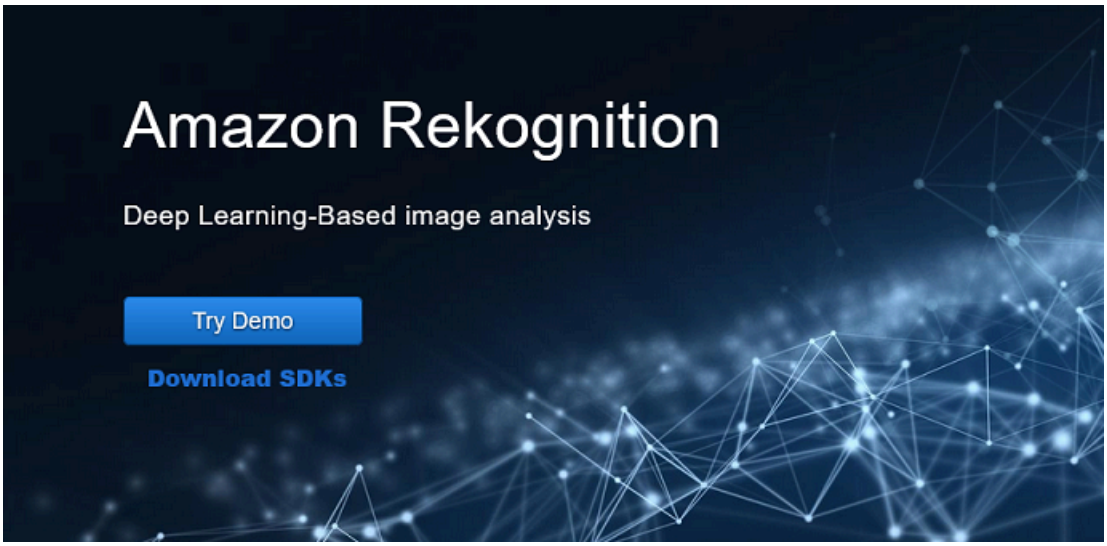
使用 Amazon Rekognition 控制台，您可以管理与 Rekognition 自定义标签和自定义审核功能相关的资源。该控制台仅提供其他 Rekognition 特征的演示。

此部分说明如何在图像集中使用 Amazon Rekognition 的一部分功能（例如，对象和场景检测、人脸分析和人脸比较）。有关更多信息，请参阅 Amazon Rekognition 的工作原理。您还可以使用 Amazon Rekognition AWS CLI API 或检测物体和场景、检测人脸以及比较和搜索人脸。有关更多信息，请参阅第 3 步：开始使用 AWS CLI 和 S AWS DK API。

本节还向您展示了如何使用 Rekognition 控制台查看 Rekognition 的汇总亚马逊 CloudWatch 指标。

主题

- 设置控制台权限
- 练习 1：检测对象和场景（控制台）
- 练习 2：分析图像中的人脸（控制台）
- 练习 3：比较图像中的人脸（控制台）
- 练习 4：查看聚合指标（控制台）



设置控制台权限

要使用 Rekognition 控制台，您需要拥有访问控制台的角色或账户的相应权限。对于某些操作，Rekognition 会自动创建一个 Amazon S3 存储桶来存储操作期间处理的文件。如果想将训练文件存储在此控制台存储桶以外的存储桶中，则需要额外的权限。

允许访问控制台

要使用 Rekognition 控制台，您可以使用如下所示的 IAM 策略，该策略涵盖 Amazon S3 和 Rekognition 控制台。有关分配权限的信息，请参阅分配权限。

```
            {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RekognitionFullAccess",
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock",
                "s3:PutCors",
                "s3:GetCors"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
        {
```

```
            "Sid": "RekognitionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
        {
            "Sid": "RekognitionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:HeadObject",
                "s3:DeleteObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*/*"
        },
        {
            "Sid": "RekognitionConsoleManifestAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:*",
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
}
```

访问外部 Amazon S3 存储桶

当你首次在新 AWS 区域中打开 Rekognition 控制台时，Rekognition 会创建一个用于存储项目文件的存储桶（控制台存储桶）。或者，您可以使用自己的 Amazon S3 存储桶（外部存储桶）将图像或清单文件上传到控制台。要使用外部存储桶，请将以下策略块添加到先前的策略中。

```
{
            "Sid": "s3ExternalBucketPolicies",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectVersion",
                "s3:GetObjectTagging",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ]
}
```

分配权限

要提供访问权限，请为您的用户、组或角色添加权限：

- AWS IAM Identity Center（Amazon Single Sign-On 的后继者）中的用户和群组：

  创建权限集合。按照《AWS IAM Identity Center（Amazon Single Sign-On 的后继者）用户指南》中的创建权限集中的指示。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

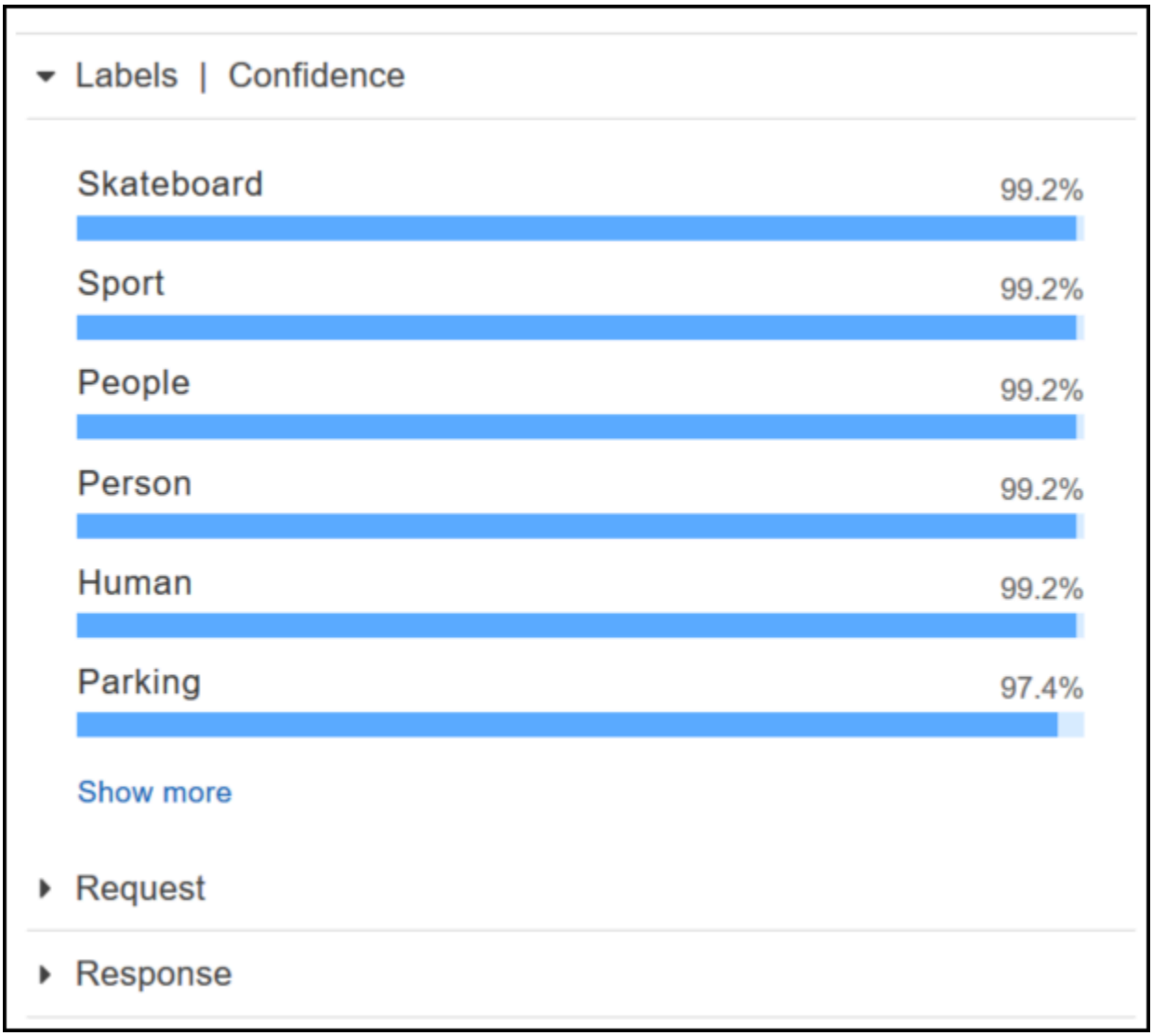
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

练习 1：检测对象和场景（控制台）

此部分简要说明 Amazon Rekognition 的对象和场景检测功能的工作方式。在您指定图像作为输入时，该服务将检测图像中的对象和场景，并将每个对象和场景连同其百分比置信度得分一起返回。

例如，Amazon Rekognition 将检测示例图像中的以下对象和场景：滑板、运动、人员和车辆。

Amazon Rekognition 还返回在示例图像中检测到的每个对象的置信度得分，如以下示例响应中所示。



要查看此响应中显示的所有置信度得分，请在标签 | 置信度窗格中选择显示更多。

您也可以查看对 API 的请求和来自 API 的响应以作为参考。

请求

```
{
   "contentString":{
      "Attributes":[
         "ALL"
      ],
      "Image":{
         "S3Object":{
            "Bucket":"console-sample-images",
```

```
            "Name":"skateboard.jpg"
         }
      }
   }
}
```

响应

```
{
   "Labels":[
      {
         "Confidence":99.25359344482422,
         "Name":"Skateboard"
      },
      {
         "Confidence":99.25359344482422,
         "Name":"Sport"
      },
      {
         "Confidence":99.24723052978516,
         "Name":"People"
      },
      {
         "Confidence":99.24723052978516,
         "Name":"Person"
      },
      {
         "Confidence":99.23908233642578,
         "Name":"Human"
      },
      {
         "Confidence":97.42484283447266,
         "Name":"Parking"
      },
      {
         "Confidence":97.42484283447266,
         "Name":"Parking Lot"
      },
      {
         "Confidence":91.53300476074219,
         "Name":"Automobile"
      },
      {
```

```
            "Confidence":91.53300476074219,
            "Name":"Car"
         },
         {
            "Confidence":91.53300476074219,
            "Name":"Vehicle"
         },
         {
            "Confidence":76.85114288330078,
            "Name":"Intersection"
         },
         {
            "Confidence":76.85114288330078,
            "Name":"Road"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Boardwalk"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Path"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Pavement"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Sidewalk"
         },
         {
            "Confidence":76.21503448486328,
            "Name":"Walkway"
         },
         {
            "Confidence":66.71541595458984,
            "Name":"Building"
         },
         {
            "Confidence":62.04711151123047,
            "Name":"Coupe"
         },
         {
```

```
            "Confidence":62.04711151123047,
            "Name":"Sports Car"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"City"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"Downtown"
         },
         {
            "Confidence":61.98909378051758,
            "Name":"Urban"
         },
         {
            "Confidence":60.978023529052734,
            "Name":"Neighborhood"
         },
         {
            "Confidence":60.978023529052734,
            "Name":"Town"
         },
         {
            "Confidence":59.22066116333008,
            "Name":"Sedan"
         },
         {
            "Confidence":56.48063278198242,
            "Name":"Street"
         },
         {
            "Confidence":54.235477447509766,
            "Name":"Housing"
         },
         {
            "Confidence":53.85226058959961,
            "Name":"Metropolis"
         },
         {
            "Confidence":52.001792907714844,
            "Name":"Office Building"
         },
         {
```

```
            "Confidence":51.325313568115234,
            "Name":"Suv"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"Apartment Building"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"High Rise"
         },
         {
            "Confidence":50.68067932128906,
            "Name":"Pedestrian"
         },
         {
            "Confidence":50.59548568725586,
            "Name":"Freeway"
         },
         {
            "Confidence":50.568580627441406,
            "Name":"Bumper"
         }
      ]
}
```


有关更多信息，请参阅 Amazon Rekognition 的工作原理。

检测您提供的图像中的对象和场景

您可以上传您拥有的图像，也可以在 Amazon Rekognition 控制台中提供图像的 URL 作为输入。Amazon Rekognition 返回对象和场景、每个对象的置信度得分以及它在您提供的图像中检测到的场景。

> **Note**
>
> 图像的大小必须小于 5MB，并且格式必须为 JPEG 或 PNG。

检测您提供的图像中的对象和场景

1. 打开亚马逊 Rekognition 控制台，网址为https://console.aws.amazon.com/rekognition/。

2. 选择标签检测。
3. 请执行以下操作之一：
   - 上传图像 – 选择上传，转到图像的存储位置，然后选择图像。
   - 使用 URL – 在文本框中键入 URL，然后选择前往。
4. 在标签 | 置信度窗格中查看检测到的每个标签的置信度得分。

有关更多图像分析选项，请参阅the section called “使用图像”。

检测您提供的视频中的对象和人物

您可以在 Amazon Rekognition 控制台中上传您作为输入内容提供的视频。Amazon Rekognition 会返回视频中检测到的人物、对象和标签。

> **Note**
>
> 演示视频不得超过一分钟或大于 30 MB。它必须采用 MP4 文件格式并使用 H.264 编解码器进行编码。

检测您提供的视频中的对象和人物

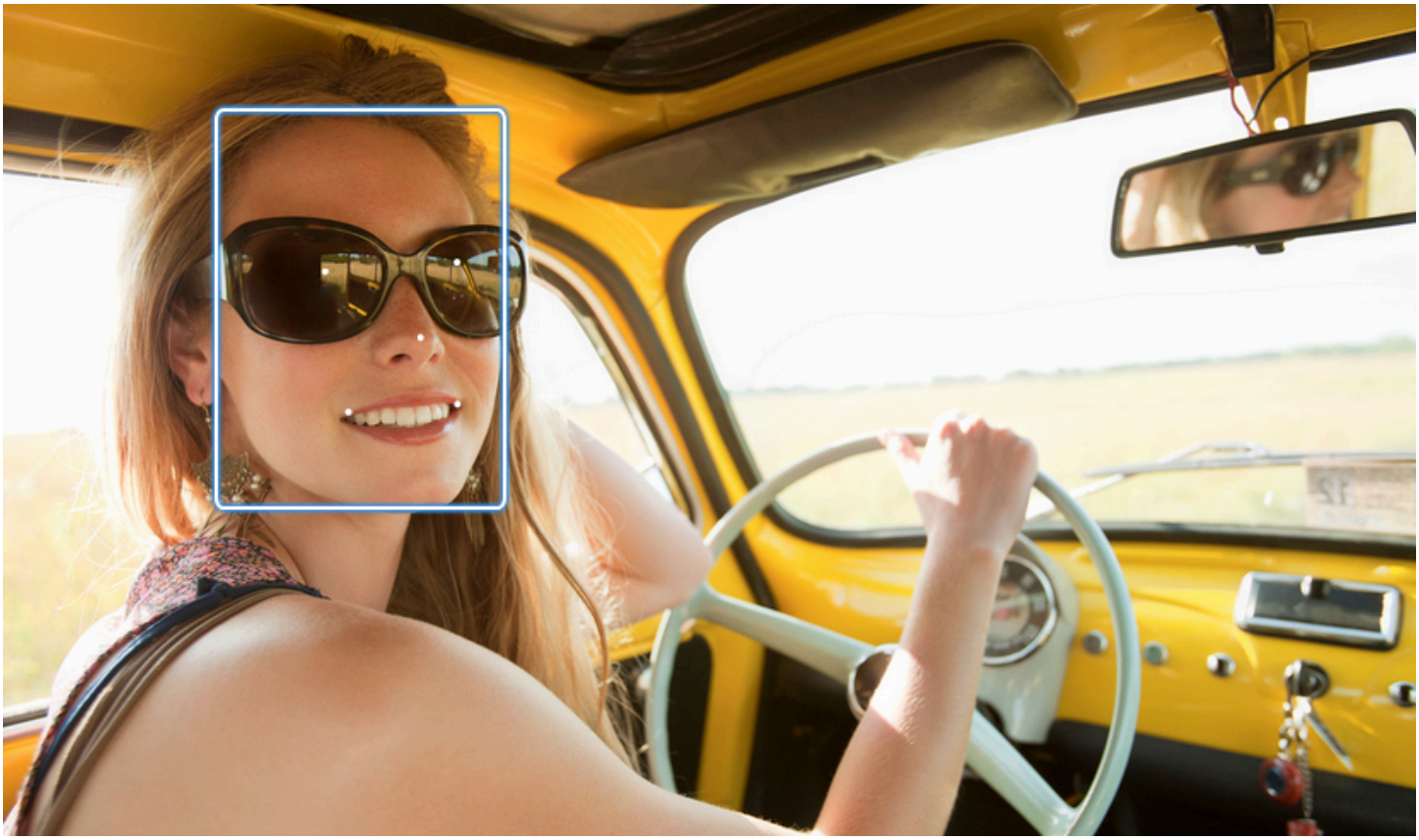
1. 打开亚马逊 Rekognition 控制台，网址为https://console.aws.amazon.com/rekognition/。
2. 从导航栏中选择存储的视频分析。
3. 在选择示例或者上载您自己的内容下，从下拉菜单中选择您自己的视频。
4. 拖放视频或从存储视频的位置选择视频。

有关更多视频分析选项，请参阅the section called “使用存储视频分析操作”或the section called “使用流视频事件”。

练习 2：分析图像中的人脸（控制台）

此部分说明如何使用 Amazon Rekognition 控制台检测图像中的人脸和分析人脸属性。如果提供的图像包含人脸作为输入，则服务将检测图像中的人脸，分析人脸的人脸属性，然后返回在图像中检测到的人脸和人脸属性的百分比置信度得分。有关更多信息，请参阅 Amazon Rekognition 的工作原理。

例如，如果您选择以下示例图像作为输入，则 Amazon Rekognition 会将它检测为人脸并返回检测到的人脸和人脸属性的置信度得分。



下面显示的是示例响应。

Results

looks like a face	99.8%
appears to be female	100%
age range	23 - 38 years old
smiling	99.4%
appears to be happy	93.2%
wearing eyeglasses	99.9%
wearing sunglasses	97.6%
eyes are open	96.2%
mouth is open	72.5%
does not have a mustache	77.6%
does not have a beard	97.1%

Show less

如果输入图像中有多个人脸，则 Rekognition 将检测图像中的最多 100 个人脸。检测到的每个人脸用一个方块标记。当您单击人脸上用方块标记的区域时，Rekognition 会在人脸 | 置信度窗格中显示检测到的人脸及其属性的置信度得分。

分析您提供的图像中的人脸

您可以上传自己的图像或在 Amazon Rekognition 控制台中提供图像的 URL。

> **Note**
>
> 图像的大小必须小于 5MB，并且格式必须为 JPEG 或 PNG。

分析您提供的图像中的人脸

1. 打开亚马逊 Rekognition 控制台，网址为https://console.aws.amazon.com/rekognition/。
2. 选择面部分析。
3. 请执行以下操作之一：
   - 上传图像 – 选择上传，转到图像的存储位置，然后选择图像。
   - 使用 URL – 在文本框中键入 URL，然后选择前往。
4. 在人脸 | 置信度窗格中查看检测到的人脸及其人脸属性的置信度得分。
5. 如果图像中有多个人脸，请选择其中一个人脸以查看其属性和分数。

练习 3：比较图像中的人脸（控制台）

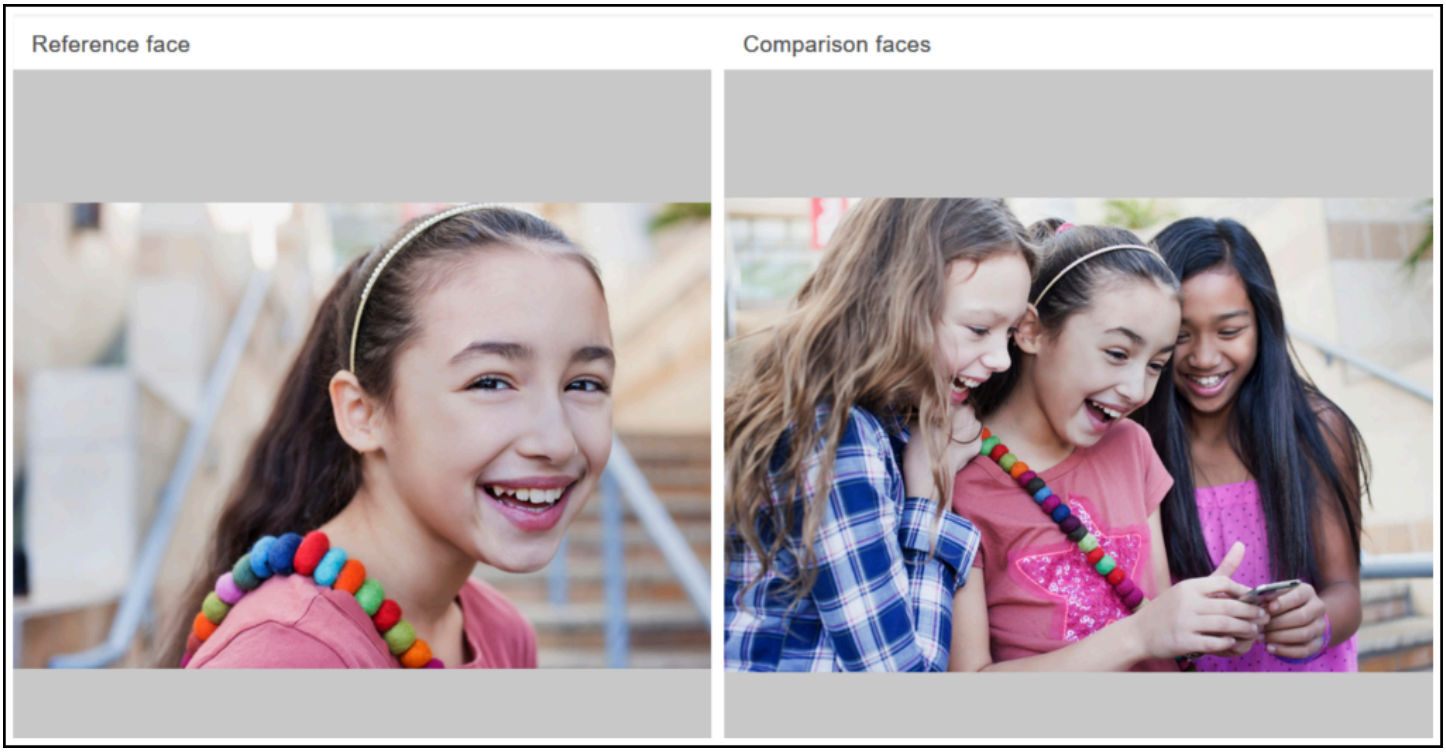
此部分说明如何使用 Amazon Rekognition 控制台将一组图像中的人脸与其中的多个人脸进行比较。在指定参考人脸（源）和比较人脸（目标）图像时，Rekognition 会将源图像中的最大人脸（即参考人脸）与目标图像中检测到的最多 100 个人脸（即比较人脸）进行比较，然后确定源中的人脸与目标图像中的人脸的接近程度。Results 窗格中将显示每次比较的相似度得分。

如果目标图像包含多个人脸，则 Rekognition 会将源图像中的人脸与在目标图像中检测到的最多 100 个图像进行匹配，然后为每个匹配分配一个相似度得分。

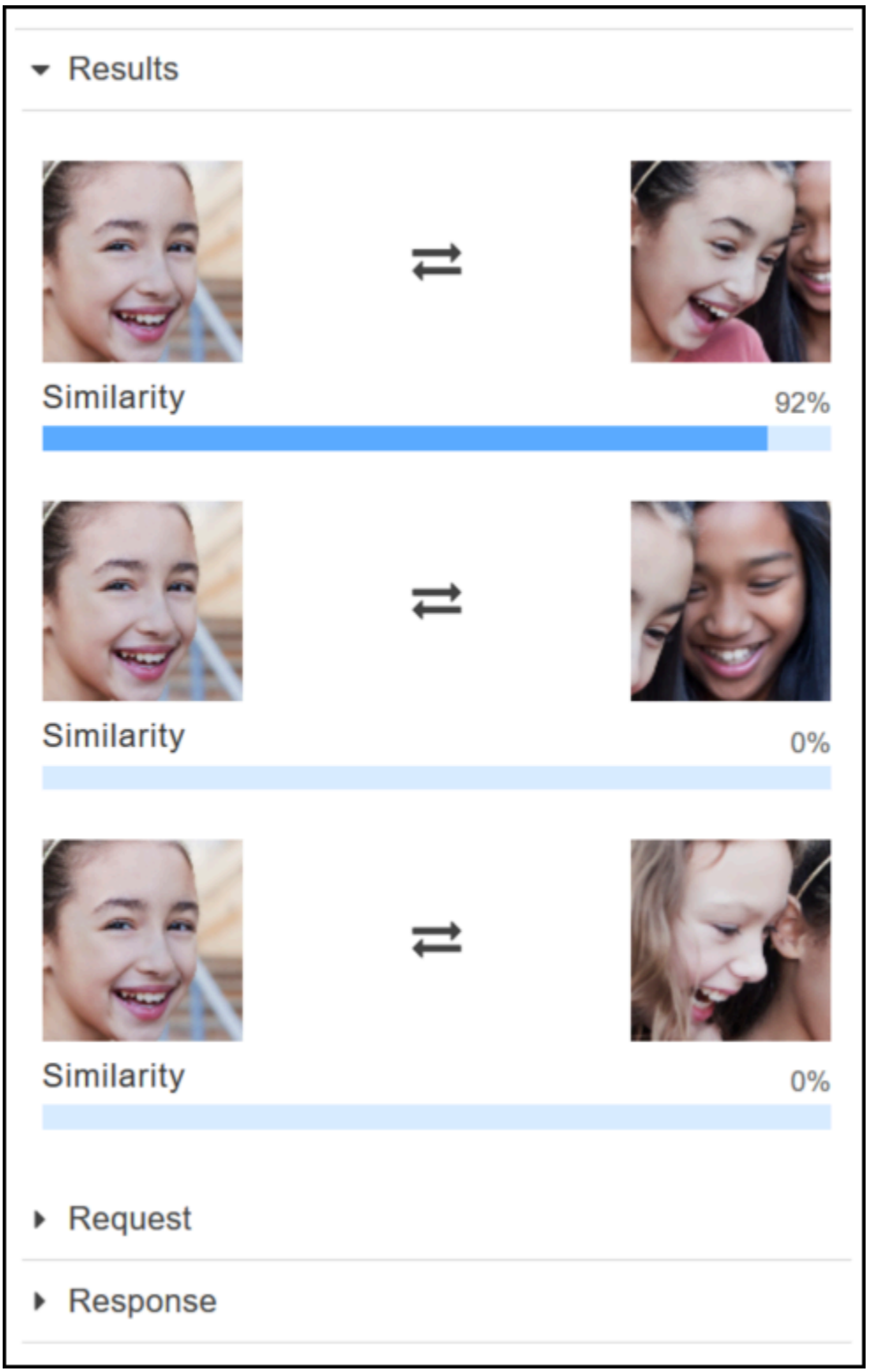
如果源图像包含多个人脸，则服务会检测源图像中的最大人脸，并使用它来与在目标图像中检测到的每个人脸进行比较。

有关更多信息，请参阅 比较图像中的人脸。

例如，如果将左侧显示的示例图像作为源图像，并将右侧的示例图像作为目标图像，Rekognition 将检测源图像中的人脸，将该人脸与在目标图像中检测到的每个人脸进行比较，并显示每个人脸对的相似度得分。



下面显示了在目标图像中检测到的人脸以及每个人脸的相似度得分。



比较您提供的图像中的人脸

您可以上传自己的源图像和目标图像以便 Rekognition 比较图像中的人脸，也可以指定图像位置的 URL。

> **Note**
>
> 图像的大小必须小于 5MB，并且格式必须为 JPEG 或 PNG。

比较图像中的人脸

1. 打开亚马逊 Rekognition 控制台，网址为https://console.aws.amazon.com/rekognition/。
2. 选择人脸比较。
3. 对于您的源图像，请执行下列操作之一：
   - 上传图像 – 选择左侧的上传，转至源图像的存储位置，然后选择图像。
   - 使用 URL – 在文本框中键入源图像的 URL，然后选择前往。
4. 对于您的目标图像，请执行下列操作之一：
   - 上传图像 – 选择右侧的上传，转至源图像的存储位置，然后选择图像。
   - 使用 URL – 在文本框中键入源图像的 URL，然后选择前往。
5. Rekognition 会将源图像中的最大人脸与目标图像中的最多 100 个人脸进行匹配，然后在结果窗格中显示每对的相似度得分。

练习 4：查看聚合指标（控制台）

Amazon Rekognition 指标窗格显示指定时段内单个 Rekognition 指标聚合的活动图表。例如，`SuccessfulRequestCount` 聚合指标显示过去 7 天对所有 Rekognition API 操作发出的成功请求的总数。

下表列出了 Rekognition 指标窗格中显示的图表以及相应的 Rekognition 指标。有关更多信息，请参阅 CloudWatchRekognition 的指标。

图表	聚合指标
成功调用数	SuccessfulRequestCount
客户端错误数	UserErrorCount
服务器错误数	ServerErrorCount

图表	聚合指标
受限	ThrottledCount
检测到的标签数	DetectedLabelCount
检测到的人脸数	DetectedFaceCount

每个图表均显示指定时段内收集的聚合指标数据。还将显示该时段的聚合指标数据的总数。要查看各个 API 调用的指标，请选择每个图表下方的链接。

要允许用户访问 Rekognition 指标窗格，请确保用户拥有相应的 Rekognition 权限。 CloudWatch例如，具有 `AmazonRekognitionReadOnlyAccess` 和 `CloudWatchReadOnlyAccess` 托管策略权限的用户可以查看指标窗格。如果某个用户没有必需的权限，则当该用户打开指标窗格时，将不会显示任何图表。有关更多信息，请参阅 适用于 Amazon Rekognition 的身份和访问管理。

有关使用监控 Rekognition 的更多信息，请参阅。 CloudWatch 使用亚马逊监控 Rekognition CloudWatch

查看聚合指标（控制台）

1. 打开亚马逊 Rekognition 控制台，网址为https://console.aws.amazon.com/rekognition/。
2. 在导航窗格中，选择指标。
3. 在下拉菜单中，选择要查看的指标所对应的时段。
4. 要更新图表，请选择刷新按钮。
5. 要查看特定聚合 CloudWatch 指标的详细指标，请选择指标图表下 CloudWatch方的查看详细信息。

使用图像和视频

您可以对三种不同类型的媒体（图像、存储视频和流视频）使用 Amazon Rekognition API 操作。本节提供有关编写代码来访问 Amazon Rekognition 以处理不同类型媒体的一般信息。有关最佳实践和注意事项的指导，请根据所处理的媒体类型，参阅下列相应章节。

本指南中的其他部分提供有关特定类型的图像和视频分析（例如人脸检测）的信息。

主题

- 使用图像
- 使用存储视频分析操作
- 使用流视频事件
- 错误处理
- 将 Amazon Rekognition 用作 FedRAMP 授权服务

使用图像

本节介绍 Amazon Rekognition Image 可对图像执行的分析类型。

- 对象和场景检测
- 人脸检测和比较
- 在集合中搜索人脸
- 名人识别
- 图像监管
- 图像文本检测

这些是通过非存储 API 操作执行的，其中 Amazon Rekognition Image 不会保留该操作所发现的任何信息。非存储 API 操作不会保留任何输入图像字节。有关更多信息，请参阅 了解非存储和存储 API 操作。

Amazon Rekognition Image 还可将脸部元数据存储在集合中以便之后检索。有关更多信息，请参阅 在集合中搜索人脸。

在此节中，使用 Amazon Rekognition Image API 操作分析存储在 Amazon S3 存储桶中的图像以及从本地文件系统中加载的图像字节。此节还介绍了通过 .jpg 图像获取图像方向信息。

Rekognition 仅使用 RGB 通道进行推理。 AWS 建议用户在使用显示器目视（由人类手动）检查比较之前移除 Alpha 通道。

主题

- 图像规格
- 分析存储在 Amazon S3 存储桶中的图像
- 分析从本地文件系统加载的图像
- 显示边界框
- 获取图像方向和边界框坐标

图像规格

Amazon Rekognition Image 操作可分析 .jpg 或 .png 格式的图像。

您将图像字节作为调用的一部分传递给 Amazon Rekognition Image 操作或引用现有 Amazon S3 对象。有关分析 Amazon S3 存储桶中所存储图像的示例，请参阅分析存储在 Amazon S3 存储桶中的图像。有关将图像字节传递到 Amazon Rekognition Image API 操作的更多信息，请参阅分析从本地文件系统加载的图像。

如果您使用 HTTP 并将图像字节作为 Amazon Rekognition Image 操作的一部分传递，则图像字节必须为 base64 编码的字符串。如果您使用 AWS 开发工具包并将图像字节作为 API 操作调用的一部分传递，则图像字节是否需要 base64 编码取决于您使用的语言。

以下常见的图像 AWS SDKs 会自动进行 base64 编码，在调用 Amazon Rekognition Image API 操作之前，您无需对图像字节进行编码。

- Java
- JavaScript
- Python
- PHP

如果您使用的是其他 AWS 软件开发工具包并在调用 Rekognition API 操作时获得图像格式错误，请尝试在将图像字节传递给 Rekognition API 操作之前对图像字节进行 base64 编码。

如果您使用调用 Amazon Rekognition Image 操作，则不支持在调用过程中传递图像字节。 AWS CLI 您必须先将图像上传到 Amazon S3 存储桶，然后再调用引用所上传图像的操作。

> **Note**
>
> 如果您传递存储在 S3Object 中的图像而不是图像字节，则图像无需 base64 编码。

有关确保 Amazon Rekognition Image 操作具有可能最低的延迟的信息，请参阅Amazon Rekognition Image 操作延迟。

校正图像方向

在多个 Rekognition API 操作中，将返回已分析图像的方向。必须知道图像方向，因为这使您能够重定图像的显示方向。分析人脸的 Rekognition API 操作还将针对人脸在图像内的位置返回边界框。您可以使用边界框在图像上人脸的周围显示一个框。返回的边界框坐标受图像方向影响，您可能需要转换边界框坐标以在人脸周围正确显示一个框。有关更多信息，请参阅 获取图像方向和边界框坐标。

调整图像大小

在分析过程中，Amazon Rekognition 会使用一组最适合特定模型或算法的预定义范围在内部调整图像大小。因此，根据输入图像的分辨率，Amazon Rekognition 可能会检测到不同数量的对象，或者提供不同的结果。例如，假设您拥有两张图片。第一张图像的分辨率为 1024x768 像素。第二张图像是第一张图像的大小调整后的版本，分辨率为 640x480 像素。如果您向提交图片 DetectLabels，则两次调用的回复DetectLabels可能会略有不同。

分析存储在 Amazon S3 存储桶中的图像

Amazon Rekognition Image 可以分析存储在 Amazon S3 存储桶中的图像或作为图像字节提供的图像。

在本主题中，您将使用 DetectLabelsAPI 操作来检测存储在 Amazon S3 存储桶中的图像（JPEG 或 PNG）中的对象、概念和场景。使用图像输入参数将图像传递给 Amazon Rekognition Image API 操作。在 Image 中，您指定 S3Object 对象属性来引用存储在 S3 存储桶中的图像。存储在 Amazon S3 存储桶中的图像的图像字节不需要 base64 编码。有关更多信息，请参阅 图像规格。

示例请求

在 DetectLabels 的示例 JSON 请求中，源图像（input.jpg）从名为 amzn-s3-demo-bucket 的 Amazon S3 存储桶加载。请注意，包含 S3 对象的 S3 存储桶的区域必须与您用于 Amazon Rekognition Image 操作的区域匹配。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75
}
```

以下示例使用各种 AWS SDKs 和 to AWS CLI call DetectLabels。有关 DetectLabels 操作响应的信息，请参阅DetectLabels 响应。

检测图像中的标签

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。确保您已为调用 API 操作的用户授予适当的编程访问权限，有关如何执行此操作的说明，请参阅授予编程式访问权限。

2. 将其中包含一个或多个对象（如树木、房屋和船）的图像上传到您的 S3 存储桶。图像的格式必须为 .jpg 或 .png。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 DetectLabels 操作。

   Java

   此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

   ```
   //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
   //PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
   amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

   package com.amazonaws.samples;
   ```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.S3Object;
import java.util.List;

public class DetectLabels {

   public static void main(String[] args) throws Exception {

      String photo = "input.jpg";
      String bucket = "bucket";


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      DetectLabelsRequest request = new DetectLabelsRequest()
           .withImage(new Image()
           .withS3Object(new S3Object()
           .withName(photo).withBucket(bucket)))
           .withMaxLabels(10)
           .withMinConfidence(75F);

      try {
         DetectLabelsResult result = rekognitionClient.detectLabels(request);
         List <Label> labels = result.getLabels();

         System.out.println("Detected labels for " + photo);
         for (Label label: labels) {
            System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
         }
      } catch(AmazonRekognitionException e) {
         e.printStackTrace();
      }
   }
}
```

AWS CLI

此示例显示 detect-labels CLI 操作的 JSON 输出。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
aws rekognition detect-labels --image '{ "S3Object": { "Bucket": "bucket-name",
 "Name": "file-name" } }' \
--features GENERAL_LABELS IMAGE_PROPERTIES \
--settings '{"ImageProperties": {"MaxDominantColors":1}, {"GeneralLabels":
{"LabelInclusionFilters":["Cat"]}}}' \
--profile profile-name \
--region us-east-1
```

如果您使用的是 Windows，则可能需要转义引号，如下例所示。

```
aws rekognition detect-labels --image "{\"S3Object\":{\"Bucket\":\"bucket-
name\",\"Name\":\"file-name\"}}" --features GENERAL_LABELS IMAGE_PROPERTIES --
settings "{\"GeneralLabels\":{\"LabelInclusionFilters\":[\"Car\"]}}" --profile
 profile-name --region us-east-1
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
```

```
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectLabels {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,ImageBucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

        try {
            S3Object s3Object = S3Object.builder()
                .bucket(bucket)
                .name(image)
```

```
                .build() ;

            Image myImage = Image.builder()
                .s3Object(s3Object)
                .build();

            DetectLabelsRequest detectLabelsRequest =
 DetectLabelsRequest.builder()
                .image(myImage)
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

Python

此示例显示在输入图像中检测到的标签。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_labels(photo, bucket):
```

```
    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
    MaxLabels=10,
    # Uncomment to use image properties and filtration settings
    #Features=["GENERAL_LABELS", "IMAGE_PROPERTIES"],
    #Settings={"GeneralLabels": {"LabelInclusionFilters":["Cat"]},
    # "ImageProperties": {"MaxDominantColors":10}}
    )

    print('Detected labels for ' + photo)
    print()
    for label in response['Labels']:
        print("Label: " + label['Name'])
        print("Confidence: " + str(label['Confidence']))
        print("Instances:")

        for instance in label['Instances']:
            print(" Bounding box")
            print(" Top: " + str(instance['BoundingBox']['Top']))
            print(" Left: " + str(instance['BoundingBox']['Left']))
            print(" Width: " + str(instance['BoundingBox']['Width']))
            print(" Height: " + str(instance['BoundingBox']['Height']))
            print(" Confidence: " + str(instance['Confidence']))
            print()

        print("Parents:")
        for parent in label['Parents']:
           print(" " + parent['Name'])

        print("Aliases:")
        for alias in label['Aliases']:
            print(" " + alias['Name'])

            print("Categories:")
        for category in label['Categories']:
            print(" " + category['Name'])
            print("----------")
            print()

    if "ImageProperties" in str(response):
        print("Background:")
```

```
            print(response["ImageProperties"]["Background"])
            print()
            print("Foreground:")
            print(response["ImageProperties"]["Foreground"])
            print()
            print("Quality:")
            print(response["ImageProperties"]["Quality"])
            print()

        return len(response['Labels'])

def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    label_count = detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))

if __name__ == "__main__":
    main()
```

Node.Js

此示例显示与图像中检测到的标签相关的信息。

将 photo 的值更改为包含一张或多张名人人脸的图像的路径和文件名。将bucket的值更改为包含所提供图像文件的 S3 存储桶的名称。将REGION的值更改为与您的账户关联的区域名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
// Import required AWS SDK clients and commands for Node.js
import { DetectLabelsCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"

// Create SNS service object.
const rekogClient = new RekognitionClient({
  region: REGION,
  credentials: fromIni({
       profile: 'profile-name',
  }),
```

```
});

const bucket = 'bucket-name'
const photo = 'photo-name'

// Set params
const params = {For example, to grant
    Image: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
  }

const detect_labels = async () => {
    try {
        const response = await rekogClient.send(new
 DetectLabelsCommand(params));
        console.log(response.Labels)
        response.Labels.forEach(label =>{
            console.log(`Confidence: ${label.Confidence}`)
            console.log(`Name: ${label.Name}`)
            console.log('Instances:')
            label.Instances.forEach(instance => {
                console.log(instance)
            })
            console.log('Parents:')
            label.Parents.forEach(name => {
                console.log(name)
            })
            console.log("-------")
        })
        return response; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
};

detect_labels();
```

.NET

此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
```

```
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Ruby

此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
  # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket = 'bucket' # the bucket name without s3://
  photo  = 'photo' # the name of file
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    image: {
      s3_object: {
        bucket: bucket,
        name: photo
      },
    },
    max_labels: 10
  }
 response = client.detect_labels attrs
 puts "Detected labels for: #{photo}"
 response.labels.each do |label|
   puts "Label:      #{label.name}"
   puts "Confidence: #{label.confidence}"
   puts "Instances:"
   label['instances'].each do |instance|
     box = instance['bounding_box']
```

```
      puts "  Bounding box:"
      puts "    Top:        #{box.top}"
      puts "    Left:       #{box.left}"
      puts "    Width:      #{box.width}"
      puts "    Height:     #{box.height}"
      puts "  Confidence: #{instance.confidence}"
    end
    puts "Parents:"
    label.parents.each do |parent|
      puts "  #{parent.name}"
    end
    puts "------------"
    puts ""
  end
```

响应示例

来自 DetectLabels 的响应是在图像中检测到的一组标签和检测标签时所依据的置信度级别。

当您对图像执行 DetectLabels 操作时，Amazon Rekognition 会返回与以下示例响应类似的输出。

响应显示操作检测到了多个标签，包括“人”、“交通工具”和“汽车”。每个标签均有一个关联的置信度级别。例如，检测算法对图像包含人的置信度为 98.991432%。

响应还包括了 Parents 数组中一个标签的原级标签。例如，标签“汽车”有两个父标签，分别名为“车辆”和“运输”。

常见对象标签的响应包括边界框信息，针对输入图像上标签的位置。例如，“人”标签有包含两个边界框的实例数组。这两个边界框是在图像中检测到的两个人的位置。

字段 LabelModelVersion 包含由 DetectLabels 使用的检测模型的版本号。

有关使用 DetectLabels 操作的更多信息，请参阅检测对象和概念。

```
{

    {
    "Labels": [
        {
            "Name": "Vehicle",
            "Confidence": 99.15271759033203,
            "Instances": [],
```

```
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Transportation",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": []
        },
        {
            "Name": "Automobile",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Car",
            "Confidence": 99.15271759033203,
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Height": 0.18528179824352264,
                        "Left": 0.0037978808395564556,
                        "Top": 0.5039216876029968
                    },
                    "Confidence": 99.15271759033203
                },
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Height": 0.21577216684818268,
                        "Left": 0.7309805154800415,
                        "Top": 0.5251884460449219
```

```
            },
            "Confidence": 99.1286392211914
        },
        {
            "BoundingBox": {
                "Width": 0.14233611524105072,
                "Height": 0.15528248250484467,
                "Left": 0.6494812965393066,
                "Top": 0.5333095788955688
            },
            "Confidence": 98.48368072509766
        },
        {
            "BoundingBox": {
                "Width": 0.11086395382881165,
                "Height": 0.10271988064050674,
                "Left": 0.10355594009160995,
                "Top": 0.5354844927787781
            },
            "Confidence": 96.45606231689453
        },
        {
            "BoundingBox": {
                "Width": 0.06254628300666809,
                "Height": 0.053911514580249786,
                "Left": 0.46083059906959534,
                "Top": 0.5573825240135193
            },
            "Confidence": 93.65448760986328
        },
        {
            "BoundingBox": {
                "Width": 0.10105438530445099,
                "Height": 0.12226245552301407,
                "Left": 0.5743985772132874,
                "Top": 0.534368634223938
            },
            "Confidence": 93.06217193603516
        },
        {
            "BoundingBox": {
                "Width": 0.056389667093753815,
                "Height": 0.17163699865341187,
                "Left": 0.9427769780158997,
```

```
                    "Top": 0.5235804319381714
                },
                "Confidence": 92.6864013671875
            },
            {
                "BoundingBox": {
                    "Width": 0.06003860384225845,
                    "Height": 0.06737709045410156,
                    "Left": 0.22409997880458832,
                    "Top": 0.5441341400146484
                },
                "Confidence": 90.4227066040039
            },
            {
                "BoundingBox": {
                    "Width": 0.02848697081208229,
                    "Height": 0.19150497019290924,
                    "Left": 0.0,
                    "Top": 0.5107086896896362
                },
                "Confidence": 86.65286254882812
            },
            {
                "BoundingBox": {
                    "Width": 0.04067881405353546,
                    "Height": 0.03428703173995018,
                    "Left": 0.316415935754776,
                    "Top": 0.5566273927688599
                },
                "Confidence": 85.36471557617188
            },
            {
                "BoundingBox": {
                    "Width": 0.043411049991846085,
                    "Height": 0.0893595889210701,
                    "Left": 0.18293385207653046,
                    "Top": 0.5394920110702515
                },
                "Confidence": 82.21705627441406
            },
            {
                "BoundingBox": {
                    "Width": 0.031183116137981415,
                    "Height": 0.03989990055561066,
```

```
                    "Left": 0.2853088080883026,
                    "Top": 0.5579366683959961
                },
                "Confidence": 81.0157470703125
            },
            {
                "BoundingBox": {
                    "Width": 0.031113790348172188,
                    "Height": 0.056484755128622055,
                    "Left": 0.2580395042896271,
                    "Top": 0.5504819750785828
                },
                "Confidence": 56.13441467285156
            },
            {
                "BoundingBox": {
                    "Width": 0.08586374670267105,
                    "Height": 0.08550430089235306,
                    "Left": 0.5128012895584106,
                    "Top": 0.5438792705535889
                },
                "Confidence": 52.37760925292969
            }
        ],
        "Parents": [
            {
                "Name": "Vehicle"
            },
            {
                "Name": "Transportation"
            }
        ]
    },
    {
        "Name": "Human",
        "Confidence": 98.9914321899414,
        "Instances": [],
        "Parents": []
    },
    {
        "Name": "Person",
        "Confidence": 98.9914321899414,
        "Instances": [
            {
```

```
                    "BoundingBox": {
                        "Width": 0.19360728561878204,
                        "Height": 0.2742200493812561,
                        "Left": 0.43734854459762573,
                        "Top": 0.35072067379951477
                    },
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Height": 0.06597328186035156,
                        "Left": 0.9155802130699158,
                        "Top": 0.5010883808135986
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Parents": []
        }
    ],
    "LabelModelVersion": "2.0"
}


}
```

分析从本地文件系统加载的图像

Amazon Rekognition Image 操作可以分析作为图像字节提供的图像或存储在 Amazon S3 存储桶中的图像。

这些主题提供如下示例：通过使用从本地文件系统加载的文件，将图像字节提供给 Amazon Rekognition Image API 操作。使用图像输入参数将图像字节传递给 Amazon Rekognition API 操作。在 `Image` 中，您指定 `Bytes` 属性以传递 base64 编码的图像字节。

使用`Bytes`输入参数传递给 Amazon Rekognition API 操作的图像字节必须为 base64 编码。这些示例使用 SDKs 的 AWS 会自动对图像进行 base64 编码。在调用 Amazon Rekognition API 操作之前，您无需对图像字节进行编码。有关更多信息，请参阅 图像规格。

在 `DetectLabels`的此示例 JSON 请求中，源图像字节在 `Bytes` 输入参数中传递。

```
{
```

```
    "Image": {
        "Bytes": "/9j/4AAQSk....."
    },
    "MaxLabels": 10,
    "MinConfidence": 77
}
```

以下示例使用各种 AWS SDKs 和 to AWS CLI call DetectLabels。有关 DetectLabels 操作响应的信息，请参阅DetectLabels 响应。

有关客户端示 JavaScript 例，请参阅使用 JavaScript。

检测本地图像中的标签

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DetectLabels 操作。

   Java

   以下 Java 示例说明如何从本地文件系统加载图像，并使用 detectLabels AWS 软件开发工具包操作来检测标签。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
```

```
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.util.IOUtils;

public class DetectLabelsLocalFile {
    public static void main(String[] args) throws Exception {
     String photo="input.jpg";


        ByteBuffer imageBytes;
        try (InputStream inputStream = new FileInputStream(new File(photo))) {
            imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
        }


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image()
                        .withBytes(imageBytes))
                .withMaxLabels(10)
                .withMinConfidence(77F);

        try {

            DetectLabelsResult result =
 rekognitionClient.detectLabels(request);
            List <Label> labels = result.getLabels();

            System.out.println("Detected labels for " + photo);
            for (Label label: labels) {
               System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
            }

        } catch (AmazonRekognitionException e) {
            e.printStackTrace();
        }

    }
}
```

Python

以下 AWS SDK for Python 示例说明如何从本地文件系统加载图像并调用 detect_labels 操作。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的路径和文件名。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_labels_local_file(photo):


    client=boto3.client('rekognition')

    with open(photo, 'rb') as image:
        response = client.detect_labels(Image={'Bytes': image.read()})

    print('Detected labels in ' + photo)
    for label in response['Labels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))

    return len(response['Labels'])

def main():
    photo='photo'

    label_count=detect_labels_local_file(photo)
    print("Labels detected: " + str(label_count))


if __name__ == "__main__":
    main()
```

.NET

以下示例说明如何从本地文件系统加载图像和使用 DetectLabels 操作来检测标签。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectLabelsLocalfile
{
    public static void Example()
    {
        String photo = "input.jpg";

        Amazon.Rekognition.Model.Image image = new
 Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
```

```
        {
            Image = image,
            MaxLabels = 10,
            MinConfidence = 77F
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

PHP

以下 AWS SDK for PHP 示例展示了如何从本地文件系统加载图像并调用 DetectFacesAPI 操作。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的路径和文件名。

```
<?php
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    require 'vendor/autoload.php';

    use Aws\Rekognition\RekognitionClient;

    $options = [
       'region'            => 'us-west-2',
        'version'           => 'latest'
    ];

    $rekognition = new RekognitionClient($options);
```

```
    // Get local image
    $photo = 'input.jpg';
    $fp_image = fopen($photo, 'r');
    $image = fread($fp_image, filesize($photo));
    fclose($fp_image);


    // Call DetectFaces
    $result = $rekognition->DetectFaces(array(
       'Image' => array(
          'Bytes' => $image,
       ),
       'Attributes' => array('ALL')
       )
    );

    // Display info for each detected person
    print 'People: Image position and estimated age' . PHP_EOL;
    for ($n=0;$n<sizeof($result['FaceDetails']); $n++){

      print 'Position: ' . $result['FaceDetails'][$n]['BoundingBox']['Left'] . "
 "
      . $result['FaceDetails'][$n]['BoundingBox']['Top']
      . PHP_EOL
      . 'Age (low): '.$result['FaceDetails'][$n]['AgeRange']['Low']
      .  PHP_EOL
      . 'Age (high): ' . $result['FaceDetails'][$n]['AgeRange']['High']
      .  PHP_EOL . PHP_EOL;
    }
?>
```

Ruby

此示例显示在输入图像中检测到的标签的列表。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的路径和文件名。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # gem 'aws-sdk-rekognition'
    require 'aws-sdk-rekognition'
    credentials = Aws::Credentials.new(
```

```
       ENV['AWS_ACCESS_KEY_ID'],
       ENV['AWS_SECRET_ACCESS_KEY']
   )
   client   = Aws::Rekognition::Client.new credentials: credentials
   photo = 'photo.jpg'
   path = File.expand_path(photo) # expand path relative to the current
directory
   file = File.read(path)
   attrs = {
     image: {
       bytes: file
     },
     max_labels: 10
   }
   response = client.detect_labels attrs
   puts "Detected labels for: #{photo}"
   response.labels.each do |label|
     puts "Label:      #{label.name}"
     puts "Confidence: #{label.confidence}"
     puts "Instances:"
     label['instances'].each do |instance|
       box = instance['bounding_box']
       puts "  Bounding box:"
       puts "    Top:        #{box.top}"
       puts "    Left:       #{box.left}"
       puts "    Width:      #{box.width}"
       puts "    Height:     #{box.height}"
       puts "  Confidence: #{instance.confidence}"
     end
     puts "Parents:"
     label.parents.each do |parent|
       puts "  #{parent.name}"
     end
     puts "------------"
     puts ""
   end
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceImage>

            Where:
                bucketName - The name of the Amazon S3 bucket where the image is
 stored
                sourceImage - The name of the image file (for example,
 pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0] ;
        String sourceImage = args[1] ;
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }
```

```
   /**
    * Detects the labels in an image stored in an Amazon S3 bucket using the
Amazon Rekognition service.
    *
    * @param rekClient     the Amazon Rekognition client used to make the
detection request
    * @param bucketName    the name of the Amazon S3 bucket where the image is
stored
    * @param sourceImage   the name of the image file to be analyzed
    */
   public static void detectImageLabels(RekognitionClient rekClient, String
bucketName, String sourceImage) {
       try {
           S3Object s3ObjectTarget = S3Object.builder()
                   .bucket(bucketName)
                   .name(sourceImage)
                   .build();

           Image souImage = Image.builder()
                   .s3Object(s3ObjectTarget)
                   .build();

           DetectLabelsRequest detectLabelsRequest =
DetectLabelsRequest.builder()
                   .image(souImage)
                   .maxLabels(10)
                   .build();

           DetectLabelsResponse labelsResponse =
rekClient.detectLabels(detectLabelsRequest);
           List<Label> labels = labelsResponse.labels();
           System.out.println("Detected labels for the given photo");
           for (Label label : labels) {
               System.out.println(label.name() + ": " +
label.confidence().toString());
           }

       } catch (RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }
```

```
}
```

使用 JavaScript

以下 JavaScript 网页示例允许用户选择图像并查看图像中检测到的人脸的估计年龄。通过调用 DetectFaces 返回估计年龄。

使用函数加载所选图像，该 JavaScript`FileReader.readAsDataURL`函数对图像进行 base64 编码。这对于在 HTML 画布上显示图像非常有用。但这意味着，在将图像字节传递给 Amazon Rekognition Image 操作之前，必须取消对它们的编码。此示例说明如何取消对加载的图像字节的编码。如果已编码的图像字节对您没有用，请改用 `FileReader.readAsArrayBuffer`，因为加载的图像是未编码的。这意味着，无需先对图像字节取消编码即可调用 Amazon Rekognition Image 操作。有关示例，请参阅使用 readAsArray缓冲区。

运行示 JavaScript 例

1. 将示例源代码加载到编辑器中。
2. 获取 Amazon Cognito 身份池标识符。有关更多信息，请参阅 获取 Amazon Cognito 身份池标识符。
3. 在示例代码的 `AnonLog` 函数中，将 `IdentityPoolIdToUse` 和 `RegionToUse` 更改为您在获取 Amazon Cognito 身份池标识符的步骤 9 中记下的值。
4. 在 `DetectFaces` 函数中，将 `RegionToUse` 更改为您在上一步中使用的值。
5. 将示例源代码另存为一个 `.html` 文件。
6. 将此文件加载到浏览器中。
7. 选择浏览...按钮，并选择包含一个或多个人脸的图像。这将显示一个表，其中包含图像中检测到的每个人脸的估计年龄。

> **Note**
>
> 以下代码示例使用两个不再属于 Amazon Cognito 的脚本。要获取这些文件，请点击.min.js 和 aws-cognito-sdk.min.js 的链接，然后将 amazon-cognito-identity每个文件中的文本另存为单独的文件。`.js`

JavaScript 示例代码

以下代码示例使用 JavaScript V2。有关 JavaScript V3 中的示例，请参阅AWS 文档 SDK 示例 GitHub 存储库中的示例。

```
<!--
Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
-->
<!DOCTYPE html>
<html>
<head>
  <script src="aws-cognito-sdk.min.js"></script>
  <script src="amazon-cognito-identity.min.js"></script>
  <script src="https://sdk.amazonaws.com/js/aws-sdk-2.16.0.min.js"></script>
  <meta charset="UTF-8">
  <title>Rekognition</title>
</head>

<body>
  <H1>Age Estimator</H1>
  <input type="file" name="fileToUpload" id="fileToUpload" accept="image/*">
  <p id="opResult"></p>
</body>
<script>

  document.getElementById("fileToUpload").addEventListener("change", function (event) {
    ProcessImage();
  }, false);

  //Calls DetectFaces API and shows estimated ages of detected faces
  function DetectFaces(imageData) {
    AWS.region = "RegionToUse";
    var rekognition = new AWS.Rekognition();
    var params = {
      Image: {
        Bytes: imageData
      },
      Attributes: [
        'ALL',
      ]
    };
    rekognition.detectFaces(params, function (err, data) {
```

```
      if (err) console.log(err, err.stack); // an error occurred
      else {
       var table = "<table><tr><th>Low</th><th>High</th></tr>";
        // show each face and build out estimated age table
        for (var i = 0; i < data.FaceDetails.length; i++) {
          table += '<tr><td>' + data.FaceDetails[i].AgeRange.Low +
            '</td><td>' + data.FaceDetails[i].AgeRange.High + '</td></tr>';
        }
        table += "</table>";
        document.getElementById("opResult").innerHTML = table;
      }
    });
  }
  //Loads selected image and unencodes image bytes for Rekognition DetectFaces API
  function ProcessImage() {
    AnonLog();
    var control = document.getElementById("fileToUpload");
    var file = control.files[0];

    // Load base64 encoded image
    var reader = new FileReader();
    reader.onload = (function (theFile) {
      return function (e) {
        var img = document.createElement('img');
        var image = null;
        img.src = e.target.result;
        var jpg = true;
        try {
          image = atob(e.target.result.split("")[1]);

        } catch (e) {
          jpg = false;
        }
        if (jpg == false) {
          try {
            image = atob(e.target.result.split("")[1]);
          } catch (e) {
            alert("Not an image file Rekognition can process");
            return;
          }
        }
        //unencode image bytes for Rekognition DetectFaces API
        var length = image.length;
        imageBytes = new ArrayBuffer(length);
```

```
        var ua = new Uint8Array(imageBytes);
        for (var i = 0; i < length; i++) {
          ua[i] = image.charCodeAt(i);
        }
        //Call Rekognition
        DetectFaces(ua);
      };
    })(file);
    reader.readAsDataURL(file);
  }
  //Provides anonymous log on to AWS services
  function AnonLog() {

    // Configure the credentials provider to use your identity pool
    AWS.config.region = 'RegionToUse'; // Region
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'IdentityPoolIdToUse',
    });
    // Make the call to obtain credentials
    AWS.config.credentials.get(function () {
      // Credentials will be available when this function is called.
      var accessKeyId = AWS.config.credentials.accessKeyId;
      var secretAccessKey = AWS.config.credentials.secretAccessKey;
      var sessionToken = AWS.config.credentials.sessionToken;
    });
  }
</script>
</html>
```

使用 readAsArray缓冲区

以下代码片段是示例代码中该ProcessImage函数的替代实现，使用 JavaScript V2。它使用 readAsArrayBuffer 加载图像并调用 DetectFaces。由于 readAsArrayBuffer 不对加载的文件进行 base64 编码，因此，在调用 Amazon Rekognition Image 操作前无需取消对图像字节的编码。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

function ProcessImage() {
    AnonLog();
    var control = document.getElementById("fileToUpload");
    var file = control.files[0];
```

```
    // Load base64 encoded image for display
    var reader = new FileReader();
    reader.onload = (function (theFile) {
      return function (e) {
        //Call Rekognition
        AWS.region = "RegionToUse";
        var rekognition = new AWS.Rekognition();
        var params = {
          Image: {
          Bytes: e.target.result
        },
        Attributes: [
        'ALL',
      ]
    };
    rekognition.detectFaces(params, function (err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else {
       var table = "<table><tr><th>Low</th><th>High</th></tr>";
        // show each face and build out estimated age table
        for (var i = 0; i < data.FaceDetails.length; i++) {
          table += '<tr><td>' + data.FaceDetails[i].AgeRange.Low +
            '</td><td>' + data.FaceDetails[i].AgeRange.High + '</td></tr>';
        }
        table += "</table>";
        document.getElementById("opResult").innerHTML = table;
      }
    });

      };
    })(file);
    reader.readAsArrayBuffer(file);
  }
```

获取 Amazon Cognito 身份池标识符

为简便起见，该示例使用一个匿名 Amazon Cognito 身份池来提供对 Amazon Rekognition Image API 的未经身份验证的访问。这可能适合您的需求。例如，您可以使用未经身份验证的访问来在用户登录前提供对网站的免费或试用访问。要提供经过身份验证的访问，请使用 Amazon Cognito 用户群体。有关更多信息，请参阅 Amazon Cognito 用户群体。

以下过程说明如何创建一个身份池来允许访问未经身份验证的身份，以及如何获取示例代码中所需的身份池标识符。

获取身份池标识符

1. 打开 Amazon Cognito 控制台。
2. 选择创建新身份池。
3. 对于身份池名称*，键入您的身份池的名称。
4. 在未经验证的身份中，选择启用未经验证的身份的访问权限。
5. 选择创建池。
6. 选择查看详细信息，并记下未经身份验证的身份的角色名称。
7. 选择允许。
8. 在平台中，选择JavaScript。
9. 在获取 AWS 凭证中，记下代码段中显示的 `AWS.config.region` 和 `IdentityPooldId` 的值。
10. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
11. 在导航窗格中，选择角色。
12. 选择您在步骤 6 中记下的角色名称。
13. 在权限选项卡中，选择附加策略。
14. 选择 AmazonRekognitionReadOnlyAccess。
15. 选择附加策略。

显示边界框

Amazon Rekognition Image 操作可以返回在图像中检测到的项目的边界框坐标。例如，DetectFaces 操作将返回在图像中检测到的每个人脸的边界框 (BoundingBox)。您可以使用边界框坐标围绕检测到的项目显示一个框。例如，下图围绕一张人脸显示一个边界框。

BoundingBox 具有以下属性：

- 高度 – 边界框的高度（以占整个图像高度的比例显示）。
- 左侧 – 边界框的左坐标（以占整个图像宽度的比例显示）。
- 顶部 – 边界框的顶部坐标（以占整个图像高度的比例显示）。
- 宽度 – 边界框的宽度（以占整个图像宽度的比例显示）。

每个 BoundingBox 属性的值介于 0 和 1 之间。每个属性值都是占整个图像宽度（Left 和 Width）或高度（Height 和 Top）的比例。例如，如果输入图像为 700 x 200 像素，而边界框的左上坐标为 350 x 50 像素，则 API 将返回 Left 值 0.5 (350/700) 和 Top 值 0.25 (50/200)。

下图显示了每个边界框属性覆盖的图像的范围。

要以正确的位置和大小显示边界框，必须将这些 BoundingBox 值乘以图像的宽度或高度（取决于你想要的值）才能得到像素值。使用像素值显示边界框。例如，上一图像的像素大小为 608（宽）x 588（高）。人脸的边界框值为：

```
BoundingBox.Left: 0.3922065
BoundingBox.Top: 0.15567766
BoundingBox.Width: 0.284666
BoundingBox.Height: 0.2930403
```

人脸边界框的位置（以像素为单位）的计算方法如下：

```
Left coordinate = BoundingBox.Left (0.3922065) * image width (608) = 238

Top coordinate = BoundingBox.Top (0.15567766) * image height (588) = 91

Face width = BoundingBox.Width (0.284666) * image width (608) = 173

Face height = BoundingBox.Height (0.2930403) * image height (588) = 172
```

您可使用这些值围绕人脸显示一个边界框。

> **Note**
>
> 图像可以通过多种方式定向。您的应用程序可能需要旋转图像才能以正确的方向显示它。边界框坐标受图像方向的影响。您可能需要先转换坐标，然后才能在正确位置显示边界框。有关更多信息，请参阅 获取图像方向和边界框坐标。

以下示例演示如何围绕通过调用 DetectFaces 检测到的人脸显示边界框。这些示例假定图像的方向为 0 度。这些示例还演示如何从 Amazon S3 存储桶下载图像。

显示边界框

1. 如果您尚未执行以下操作，请：

a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DetectFaces 操作。

Java

将bucket的值更改为包含图像文件的 Amazon S3 存储桶。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的文件名。

```
//Loads images, detects faces and draws bounding boxes.Determines exif
 orientation, if necessary.
package com.amazonaws.samples;

//Import the basic graphics classes.
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;

import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.FaceDetail;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

// Calls DetectFaces and displays a bounding box around each detected image.
public class DisplayFaces extends JPanel {

    private static final long serialVersionUID = 1L;
```

```
   BufferedImage image;
   static int scale;
   DetectFacesResult result;

   public DisplayFaces(DetectFacesResult facesResult, BufferedImage bufImage)
throws Exception {
       super();
       scale = 1; // increase to shrink image size.

       result = facesResult;
       image = bufImage;


   }
   // Draws the bounding box around the detected faces.
   public void paintComponent(Graphics g) {
       float left = 0;
       float top = 0;
       int height = image.getHeight(this);
       int width = image.getWidth(this);

       Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

       // Draw the image.
       g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
       g2d.setColor(new Color(0, 212, 0));

       // Iterate through faces and display bounding boxes.
       List<FaceDetail> faceDetails = result.getFaceDetails();
       for (FaceDetail face : faceDetails) {

           BoundingBox box = face.getBoundingBox();
           left = width * box.getLeft();
           top = height * box.getTop();
           g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                   Math.round((width * box.getWidth()) / scale),
Math.round((height * box.getHeight())) / scale);

       }
   }


   public static void main(String arg[]) throws Exception {
```

```
        String photo = "photo.png";
        String bucket = "bucket";
        int height = 0;
        int width = 0;

        // Get the image from an S3 Bucket
        AmazonS3 s3client = AmazonS3ClientBuilder.defaultClient();

        com.amazonaws.services.s3.model.S3Object s3object =
s3client.getObject(bucket, photo);
        S3ObjectInputStream inputStream = s3object.getObjectContent();
        BufferedImage image = ImageIO.read(inputStream);
        DetectFacesRequest request = new DetectFacesRequest()
                .withImage(new Image().withS3Object(new
S3Object().withName(photo).withBucket(bucket)));

        width = image.getWidth();
        height = image.getHeight();

        // Call DetectFaces
        AmazonRekognition amazonRekognition =
AmazonRekognitionClientBuilder.defaultClient();
        DetectFacesResult result = amazonRekognition.detectFaces(request);

        //Show the bounding box info for each face.
        List<FaceDetail> faceDetails = result.getFaceDetails();
        for (FaceDetail face : faceDetails) {

            BoundingBox box = face.getBoundingBox();
            float left = width * box.getLeft();
            float top = height * box.getTop();
            System.out.println("Face:");

            System.out.println("Left: " + String.valueOf((int) left));
            System.out.println("Top: " + String.valueOf((int) top));
            System.out.println("Face Width: " + String.valueOf((int) (width *
box.getWidth())));
            System.out.println("Face Height: " + String.valueOf((int) (height *
box.getHeight())));
            System.out.println();

        }

        // Create frame and panel.
```

```
        JFrame frame = new JFrame("RotateImage");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DisplayFaces panel = new DisplayFaces(result, image);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

Python

将bucket的值更改为包含图像文件的 Amazon S3 存储桶。将 photo 的值更改为图像文件（.jpg 或 .png 格式）的文件名。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
import boto3
import io
from PIL import Image, ImageDraw

def show_faces(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    # Load image from S3 bucket
    s3_connection = boto3.resource('s3')
    s3_object = s3_connection.Object(bucket, photo)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image = Image.open(stream)

    # Call DetectFaces
    response = client.detect_faces(Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}},
                                   Attributes=['ALL'])

    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)
```

```
    # calculate and display bounding boxes for each detected face
    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']
['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        box = faceDetail['BoundingBox']
        left = imgWidth * box['Left']
        top = imgHeight * box['Top']
        width = imgWidth * box['Width']
        height = imgHeight * box['Height']

        print('Left: ' + '{0:.0f}'.format(left))
        print('Top: ' + '{0:.0f}'.format(top))
        print('Face Width: ' + "{0:.0f}".format(width))
        print('Face Height: ' + "{0:.0f}".format(height))

        points = (
            (left, top),
            (left + width, top),
            (left + width, top + height),
            (left, top + height),
            (left, top)

        )
        draw.line(points, fill='#00d400', width=2)

        # Alternatively can draw rectangle. However you can't set line width.
        # draw.rectangle([left,top, left + width, top + height],
 outline='#00d400')

    image.show()

    return len(response['FaceDetails'])

def main():
    bucket = "bucket-name"
    photo = "photo-name"
    faces_count = show_faces(photo, bucket)
    print("faces detected: " + str(faces_count))

if __name__ == "__main__":
```

```
    main()
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

请注意，s3 指的是 AWS 软件开发工具包 Amazon S3 客户端，rekClient 指的是 AWS 软件开发工具包 Amazon Rekognition 客户端。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
//snippet-end:[rekognition.java2.detect_labels.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DisplayFaces extends JPanel {

    static DetectFacesResponse result;
    static BufferedImage image;
    static int scale;

    public static void main(String[] args) throws Exception {

        final String usage = "\n" +
            "Usage: " +
            "   <sourceImage> <bucketName>\n\n" +
            "Where:\n" +
            "   sourceImage - The name of the image in an Amazon S3 bucket (for
 example, people.png). \n\n" +
            "   bucketName - The name of the Amazon S3 bucket (for example,
 amzn-s3-demo-bucket). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        String bucketName = args[1];
        Region region = Region.US_EAST_1;
        S3Client s3 = S3Client.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        displayAllFaces(s3, rekClient, sourceImage, bucketName);
        s3.close();
        rekClient.close();
    }
```

```
   // snippet-start:[rekognition.java2.display_faces.main]
   public static void displayAllFaces(S3Client s3,
                                      RekognitionClient rekClient,
                                      String sourceImage,
                                      String bucketName) {
       int height;
       int width;
       byte[] data = getObjectBytes (s3, bucketName, sourceImage);
       InputStream is = new ByteArrayInputStream(data);

       try {
           SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
           image = ImageIO.read(sourceBytes.asInputStream());
           width = image.getWidth();
           height = image.getHeight();

           // Create an Image object for the source image
           software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
               .bytes(sourceBytes)
               .build();

           DetectFacesRequest facesRequest = DetectFacesRequest.builder()
               .attributes(Attribute.ALL)
               .image(souImage)
               .build();

           result = rekClient.detectFaces(facesRequest);

           // Show the bounding box info for each face.
           List<FaceDetail> faceDetails = result.faceDetails();
           for (FaceDetail face : faceDetails) {
               BoundingBox box = face.boundingBox();
               float left = width * box.left();
               float top = height * box.top();
               System.out.println("Face:");

               System.out.println("Left: " + (int) left);
               System.out.println("Top: " + (int) top);
               System.out.println("Face Width: " + (int) (width *
box.width()));
               System.out.println("Face Height: " + (int) (height *
box.height()));
```

```
                System.out.println();
            }

            // Create the frame and panel.
            JFrame frame = new JFrame("RotateImage");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            DisplayFaces panel = new DisplayFaces(image);
            panel.setPreferredSize(new Dimension(image.getWidth() / scale,
image.getHeight() / scale));
            frame.setContentPane(panel);
            frame.pack();
            frame.setVisible(true);

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        } catch (IOException e) {
           e.printStackTrace();
        }
    }

    public static byte[] getObjectBytes (S3Client s3, String bucketName, String
keyName) {

        try {
            GetObjectRequest objectRequest = GetObjectRequest
                .builder()
                .key(keyName)
                .bucket(bucketName)
                .build();

            ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
            return objectBytes.asByteArray();

        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public DisplayFaces(BufferedImage bufImage) {
        super();
```

```
            scale = 1; // increase to shrink image size.
            image = bufImage;
        }

        // Draws the bounding box around the detected faces.
        public void paintComponent(Graphics g) {
            float left;
            float top;
            int height = image.getHeight(this);
            int width = image.getWidth(this);
            Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

            // Draw the image
            g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
            g2d.setColor(new Color(0, 212, 0));

            // Iterate through the faces and display bounding boxes.
            List<FaceDetail> faceDetails = result.faceDetails();
            for (FaceDetail face : faceDetails) {
                BoundingBox box = face.boundingBox();
                left = width * box.left();
                top = height * box.top();
                g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                        Math.round((width * box.width()) / scale),
     Math.round((height * box.height())) / scale);
            }
        }
        // snippet-end:[rekognition.java2.display_faces.main]
    }
```

获取图像方向和边界框坐标

使用 Amazon Rekognition Image 的应用程序通常需要显示 Amazon Rekognition Image 操作检测到的图像并在检测到的人脸周围显示框。要在您的应用程序中正确显示图像，您需要知道图像的方向。您可能需要校正此方向。对于某些 .jpg 文件，图像的方向包含在图像的可交换图像文件格式 (Exif) 元数据中。

要在人脸周围显示一个框，您需要人脸边界框的坐标。如果该框的方向不正确，则可能需要调整这些坐标。Amazon Rekognition Image 人脸检测操作会返回每个检测到的人脸的边界框坐标，但它不会估计没有 Exif 元数据的 .jpg 文件的坐标。

以下示例演示如何获取图像中检测到的人脸的边界框坐标

使用此示例中的信息确保您的图像定向准确以及边界框显示在您的应用程序中的正确位置。

由于用于旋转并显示图像和边界框的代码取决于您使用的语言和环境，因此我们不会在代码中解释如何显示图像和边界框，或如何从 Exif 元数据获取方向信息。

查找图像的方向

要在您的应用程序中正确显示图像，您可能需要旋转图像。下图朝向 0 度并正确显示。

但是，下图将逆时针旋转 90 度。要正确显示它，您需要找到图像的方向并使用代码中的信息将图像旋转到 0 度。

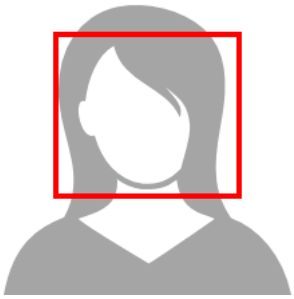
一些 .jpg 格式的图像的方向信息包含在 Exif 元数据中。如果可用，则图像的 Exif 元数据包含方向。在 Exif 元数据中，您可以在 `orientation` 字段中找到图像的方向。虽然 Amazon Rekognition Image 会发现 Exif 元数据中存在图像方向信息，但它不会提供对该信息的访问权限。要访问图像中的 Exif 元数据，请使用第三方库或编写您自己的代码。有关更多信息，请参阅 Exif 版本 2.32。

如果您知道图像的方向，可编写代码来旋转并正确显示它。

显示边界框

分析图像中人脸的 Amazon Rekognition Image 操作还将返回人脸周围的边界框的坐标。有关更多信息，请参阅 BoundingBox。

要在您的应用程序中围绕人脸显示与下图中所示的框类似的边界框，请在代码中使用边界框坐标。操作返回的边界框坐标反应图像的方向。如果您必须旋转图像才能正确显示它，您可能需要转换边界框坐标。



在 Exif 元数据中存在方向信息时显示边界框

如果图像的方向包含在 Exif 元数据中，则 Amazon Rekognition Image 操作将执行下列操作：

- 在操作响应的方向校正字段中返回 null。要旋转图像，请在代码中使用 Exif 元数据中提供的方向。
- 返回已朝向 0 度的边界框坐标。要将边界框显示在正确位置，请使用返回的坐标。您无需转换它们。

示例：获取图像的图像方向和边界框坐标

以下示例说明如何使用 AWS SDK 获取 Exif 图像方向数据和通过 `RecognizeCelebrities` 操作检测到的名人的边界框坐标。

> **Note**
>
> 自 2021 年 8 月起，已停止支持使用 `OrientationCorrection` 字段估算图像方向。API 响应中包含的该字段的任何返回值将始终为 NULL。

Java

此示例将从本地文件系统加载图像，调用 `RecognizeCelebrities` 操作，确定图像的高度和宽度并计算已旋转图像的人脸的边界框坐标。此示例未演示如何处理存储在 Exif 元数据中的方向信息。

在函数 `main` 中，请将 `photo` 的值替换为存储在本地的 .png 或 .jpg 格式的图像的名称和路径。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import javax.imageio.ImageIO;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import com.amazonaws.util.IOUtils;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.ComparedFace;

public class RotateImage {

public static void main(String[] args) throws Exception {

  String photo = "photo.png";

  //Get Rekognition client
 AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


  // Load image
  ByteBuffer imageBytes=null;
  BufferedImage image = null;

  try (InputStream inputStream = new FileInputStream(new File(photo))) {
     imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
```

```
  }
  catch(Exception e)
  {
      System.out.println("Failed to load file " + photo);
      System.exit(1);
  }

  //Get image width and height
  InputStream imageBytesStream;
  imageBytesStream = new ByteArrayInputStream(imageBytes.array());

  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  image=ImageIO.read(imageBytesStream);
  ImageIO.write(image, "jpg", baos);

  int height = image.getHeight();
  int width = image.getWidth();

  System.out.println("Image Information:");
  System.out.println(photo);
  System.out.println("Image Height: " + Integer.toString(height));
  System.out.println("Image Width: " + Integer.toString(width));

  //Call GetCelebrities

  try{
    RecognizeCelebritiesRequest request = new RecognizeCelebritiesRequest()
            .withImage(new Image()
               .withBytes((imageBytes)));


      RecognizeCelebritiesResult result =
 amazonRekognition.recognizeCelebrities(request);
      // The returned value of OrientationCorrection will always be null
      System.out.println("Orientation: " + result.getOrientationCorrection() +
 "\n");
      List <Celebrity> celebs = result.getCelebrityFaces();

      for (Celebrity celebrity: celebs) {
          System.out.println("Celebrity recognized: " + celebrity.getName());
          System.out.println("Celebrity ID: " + celebrity.getId());
          ComparedFace  face = celebrity.getFace()
;             ShowBoundingBoxPositions(height,
                  width,
```

```
                    face.getBoundingBox(),
                    result.getOrientationCorrection());

            System.out.println();
        }

    } catch (AmazonRekognitionException e) {
        e.printStackTrace();
    }

}


public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
 BoundingBox box, String rotation) {

  float left = 0;
  float top = 0;

  if(rotation==null){
      System.out.println("No estimated estimated orientation. Check Exif data.");
      return;
  }
  //Calculate face position based on image orientation.
  switch (rotation) {
     case "ROTATE_0":
        left = imageWidth * box.getLeft();
        top = imageHeight * box.getTop();
        break;
     case "ROTATE_90":
        left = imageHeight * (1 - (box.getTop() + box.getHeight()));
        top = imageWidth * box.getLeft();
        break;
     case "ROTATE_180":
        left = imageWidth - (imageWidth * (box.getLeft() + box.getWidth()));
        top = imageHeight * (1 - (box.getTop() + box.getHeight()));
        break;
     case "ROTATE_270":
        left = imageHeight * box.getTop();
        top = imageWidth * (1 - box.getLeft() - box.getWidth());
        break;
     default:
        System.out.println("No estimated orientation information. Check Exif
 data.");
```

```
        return;
  }

  //Display face location information.
  System.out.println("Left: " + String.valueOf((int) left));
  System.out.println("Top: " + String.valueOf((int) top));
  System.out.println("Face Width: " + String.valueOf((int)(imageWidth *
 box.getWidth())));
  System.out.println("Face Height: " + String.valueOf((int)(imageHeight *
 box.getHeight())));

  }
}
```

Python

此示例使用 PIL/Pillow 图像库来获取图像宽度和高度。有关更多信息，请参阅 Pillow。此示例将保留 exif 元数据，您在应用程序中的其他地方可能需要这些元数据。

在函数 `main` 中，请将 `photo` 的值替换为存储在本地的 .png 或 .jpg 格式的图像的名称和路径。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import io
from PIL import Image


# Calculate positions from from estimated rotation
def show_bounding_box_positions(imageHeight, imageWidth, box):
    left = 0
    top = 0

    print('Left: ' + '{0:.0f}'.format(left))
    print('Top: ' + '{0:.0f}'.format(top))
    print('Face Width: ' + "{0:.0f}".format(imageWidth * box['Width']))
    print('Face Height: ' + "{0:.0f}".format(imageHeight * box['Height']))


def celebrity_image_information(photo):
    client = boto3.client('rekognition')
```

```
    # Get image width and height
    image = Image.open(open(photo, 'rb'))
    width, height = image.size

    print('Image information: ')
    print(photo)
    print('Image Height: ' + str(height))
    print('Image Width: ' + str(width))

    # call detect faces and show face age and placement
    # if found, preserve exif info
    stream = io.BytesIO()
    if 'exif' in image.info:
        exif = image.info['exif']
        image.save(stream, format=image.format, exif=exif)
    else:
        image.save(stream, format=image.format)
    image_binary = stream.getvalue()

    response = client.recognize_celebrities(Image={'Bytes': image_binary})

    print()
    print('Detected celebrities for ' + photo)

    for celebrity in response['CelebrityFaces']:
        print('Name: ' + celebrity['Name'])
        print('Id: ' + celebrity['Id'])

        # Value of "orientation correction" will always be null
        if 'OrientationCorrection' in response:
            show_bounding_box_positions(height, width, celebrity['Face']
['BoundingBox'])

        print()
    return len(response['CelebrityFaces'])


def main():
    photo = 'photo'

    celebrity_count = celebrity_image_information(photo)
    print("celebrities detected: " + str(celebrity_count))
```

```
if __name__ == "__main__":
    main()
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.Celebrity;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class RotateImage {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>

                Where:
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;
```

```
        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Locating celebrities in " + sourceImage);
        recognizeAllCelebrities(rekClient, sourceImage);
        rekClient.close();
    }

    public static void recognizeAllCelebrities(RekognitionClient rekClient, String
sourceImage) {
        try {
            BufferedImage image;
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            image = ImageIO.read(sourceBytes.asInputStream());
            int height = image.getHeight();
            int width = image.getWidth();

            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            RecognizeCelebritiesRequest request =
RecognizeCelebritiesRequest.builder()
                    .image(souImage)
                    .build();

            RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request);
            List<Celebrity> celebs = result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
            for (Celebrity celebrity : celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
                System.out.println("Celebrity ID: " + celebrity.id());
                ComparedFace face = celebrity.face();
```

```
                ShowBoundingBoxPositions(height,
                        width,
                        face.boundingBox(),
                        result.orientationCorrectionAsString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
BoundingBox box, String rotation) {
        float left;
        float top;
        if (rotation == null) {
            System.out.println("No estimated estimated orientation.");
            return;
        }

        // Calculate face position based on the image orientation.
        switch (rotation) {
            case "ROTATE_0" -> {
                left = imageWidth * box.left();
                top = imageHeight * box.top();
            }
            case "ROTATE_90" -> {
                left = imageHeight * (1 - (box.top() + box.height()));
                top = imageWidth * box.left();
            }
            case "ROTATE_180" -> {
                left = imageWidth - (imageWidth * (box.left() + box.width()));
                top = imageHeight * (1 - (box.top() + box.height()));
            }
            case "ROTATE_270" -> {
                left = imageHeight * box.top();
                top = imageWidth * (1 - box.left() - box.width());
            }
            default -> {
                System.out.println("No estimated orientation information. Check Exif
data.");
```

```
                return;
            }
        }

        System.out.println("Left: " + (int) left);
        System.out.println("Top: " + (int) top);
        System.out.println("Face Width: " + (int) (imageWidth * box.width()));
        System.out.println("Face Height: " + (int) (imageHeight * box.height()));
    }
}
```

使用存储视频分析操作

Amazon Rekognition Video 是可用于分析视频的 API。利用 Amazon Rekognition Video，您可以检测存储于 Amazon Simple Storage Service (Amazon S3) 存储桶的视频中的标签、人脸、人员、名人和成人（暗示性和明显的）内容。您可以在媒体/娱乐和公共安全等类别中使用 Amazon Rekognition Video。以前，扫描视频中的物体或人员可能需要人进行数小时的查看，并且这种方式容易出错。Amazon Rekognition Video 将自动检测视频中的项目和它们出现的时间。

本节介绍 Amazon Rekognition Video 可执行的分析类型、API 的概述以及使用 Amazon Rekognition Video 的示例。

主题

- 分析类型
- Amazon Rekognition Video API 概述
- 调用 Amazon Rekognition Video 操作
- 配置 Amazon Rekognition Video
- 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)
- 使用分析视频 AWS Command Line Interface
- 参考：视频分析结果通知
- Amazon Rekognition Video 故障排除

分析类型

您可以使用 Amazon Rekognition Video 分析视频中的以下信息：

- 视频分段

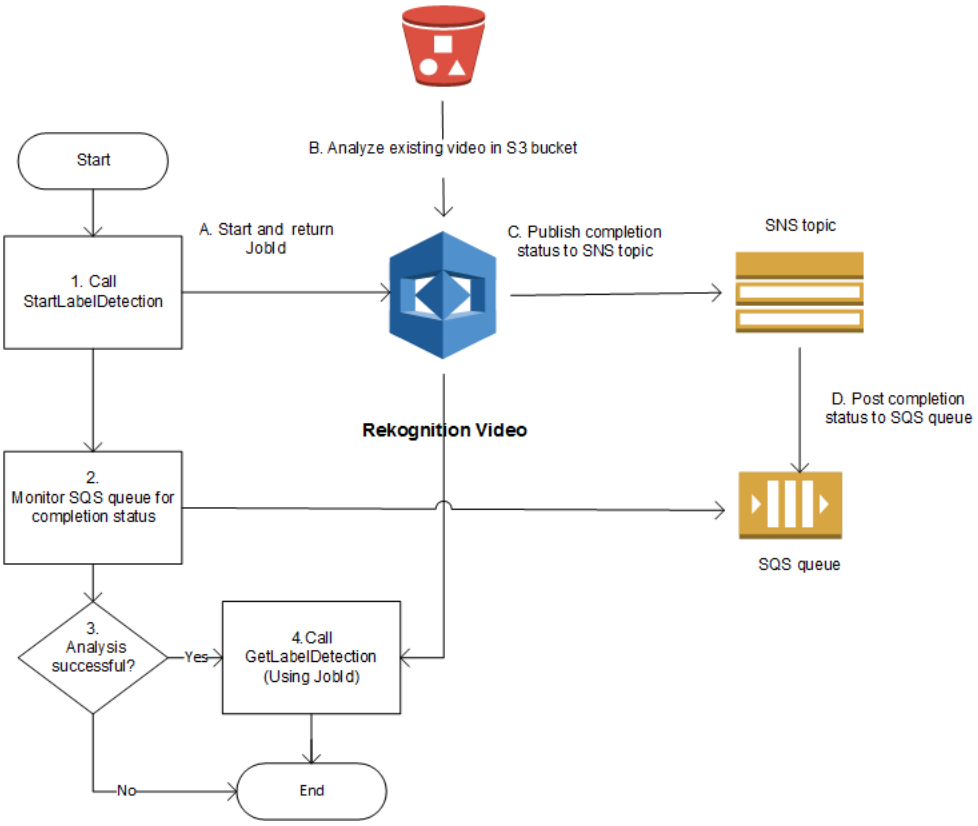
- 标签
- 暗示性和明显的成人内容
- 文本
- 名人
- 人脸
- 人员

有关更多信息，请参阅 Amazon Rekognition 的工作原理。

Amazon Rekognition Video API 概述

Amazon Rekognition Video 处理存储在 Amazon S3 存储桶中的视频。设计模式是异步操作集。您可通过调用 `Start` 操作 (如 StartLabelDetection) 来启动视频分析。将请求完成状态发布到Amazon Simple Notification Service (Amazon SNS) 主题。要从亚马逊 SNS 主题中获取完成状态，您可以使用亚马逊简单队列服务 (Amazon SQS) Simple Queue 队列或函数。 AWS Lambda 在获得完成状态之后，请调用 `Get` 操作 (如 GetLabelDetection) 以获取请求的结果。

下图显示了检测存储于 Amazon S3 存储桶的视频中的标签的过程。在此图中，Amazon SQS 队列将从 Amazon SNS 主题获取完成状态。或者，你可以使用一个 AWS Lambda 函数。



其他 Amazon Rekognition Video 操作的过程相同。下表列出了每个非存储 Amazon Rekognition 操作的 `Start` 和 `Get` 操作。

检测	Start 操作	Get 操作
视频分段	StartSegmentDetection	GetSegmentDetection
标签	StartLabelDetection	GetLabelDetection
明显的或暗示性成人内容	StartContentModeration	GetContentModeration
文本	StartTextDetection	GetTextDetection
名人	StartCelebrityRecognition	GetCelebrityRecognition
人脸	StartFaceDetection	GetFaceDetection
人员	StartPersonTracking	GetPersonTracking

对于 `Get` 之外的 `GetCelebrityRecognition` 操作，Amazon Rekognition Video 将返回有关何时在整个输入视频中检测到实体的跟踪信息。

有关使用 Amazon Rekognition Video 的更多信息，请参阅调用 Amazon Rekognition Video 操作。有关使用 Amazon SQS 执行视频分析的示例，请参阅使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。有关 AWS CLI 示例，请参见使用分析视频 AWS Command Line Interface。

视频格式和存储

Amazon Rekognition 操作可以分析存储在 Amazon S3 存储桶中的视频。有关对视频分析操作的所有限制的列表，请参阅准则和配额。

视频必须使用 H.264 编解码器进行编码。支持的文件格式为 MPEG-4 和 MOV。

编解码器是一种软件或硬件，用于压缩数据以提高传输速度以及将收到的数据解压为其原始形式。H.264 编解码器通常用于录制、压缩和分发视频内容。视频文件格式可包含一个或多个编解码器。如果您的 MOV 或 MPEG-4 格式的视频文件不适用于 Amazon Rekognition Video，请检查用于对该视频进行编码的编解码器是否为 H.264。

任何分析音频数据的 Amazon Rekognition Video API 仅支持 AAC 音频编解码器。

存储视频的最大文件大小为 10GB。

搜索人员

您可以使用存储在集合中的面部元数据来搜索视频中的人员。例如，您可以在已存档视频中搜索某个特定人员或搜索多个人员。您使用 IndexFaces 操作将源图像中的面部元数据存储在集合中。然后，您可以使用 StartFaceSearch 在集合中开始异步搜索人脸。您使用 GetFaceSearch 获取搜索结果。有关更多信息，请参阅 搜索存储视频中的人脸。搜索人员是基于存储的 Amazon Rekognition 操作的一个示例。有关更多信息，请参阅 基于存储的 API 操作。

您还可以在流视频中搜索人员。有关更多信息，请参阅 使用流视频事件。

调用 Amazon Rekognition Video 操作

Amazon Rekognition Video 是一个异步 API，可用于分析存储在 Amazon Simple Storage Service (Amazon S3) 存储桶的视频。您可以通过调用 Amazon Rekognit `Start` ion Video 操作开始分析视频，例如。StartPersonTrackingAmazon Rekognition Video 将分析请求的结果发布到 Amazon Simple Notification Service (Amazon SNS) 主题。您可以使用亚马逊简单队列服务 (Amazon SQS) Simple Queue 队列或 AWS Lambda 函数来获取来自亚马逊 SNS 主题的视频分析请求的完成状态。最后，您可以通过调用 Amazon Rekognit `Get` ion 操作来获取视频分析请求结果，例如。GetPersonTracking

以下各节中的信息使用标签检测操作来展示 Amazon Rekognition Video 如何在存储于 Amazon S3 存储桶的视频中检测标签（对象、事件、概念和活动）。同样的方法也适用于其他 Amazon Rekognition Video 操作——例如，和。StartFaceDetectionStartPersonTracking示例 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 说明了如何通过使用 Amazon SQS 队列从 Amazon SNS 主题获取完成状态来分析视频。它还用作其他 Amazon Rekognition Video 示例（如人物的轨迹）的基础。有关 AWS CLI 示例，请参见使用分析视频 AWS Command Line Interface。

主题

- 启动视频分析
- 获取 Amazon Rekognition Video 分析请求的完成状态
- 获取 Amazon Rekognition Video 分析结果

启动视频分析

您可以通过致电来启动亚马逊 Rekognition Video 标签检测请求。StartLabelDetection下面是由 `StartLabelDetection` 传递的 JSON 请求的示例。

```
{
    "Video": {
```

```
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video.mp4"
        }
    },
    "ClientRequestToken": "LabelDetectionToken",
    "MinConfidence": 50,
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnn:topic",
        "RoleArn": "arn:aws:iam::nnnnnnnnnn:role/roleopic"
    },
    "JobTag": "DetectingLabels"
}
```

输入参数 Video 提供视频文件名以及从中检索此文件的 Amazon S3 存储桶。NotificationChannel 包含 Amazon Rekognition Video 在视频分析请求完成时通知的 Amazon SNS 主题的 Amazon 资源名称 (ARN)。Amazon SNS 主题必须与您调用的 Amazon Rekognition Video 端点位于同一 AWS 区域。NotificationChannel 还包含允许 Amazon Rekognition Video 向 Amazon SNS 主题进行发布的角色的 ARN。您通过创建 IAM 服务角色为 Amazon Rekognition 提供向您的 Amazon SNS 主题进行发布的权限。有关更多信息，请参阅 配置 Amazon Rekognition Video。

您也可以指定一个可选输入参数 JobTag，该参数使您能够标识处于已发布到 Amazon SNS 主题的完成状态的任务。

为防止分析任务意外重复，您可以选择性地提供幂等令牌 ClientRequestToken。如果您为 ClientRequestToken 提供了一个值，Start 操作将为对 start 操作的多个相同调用（如 StartLabelDetection）返回相同的 JobId。ClientRequestToken 令牌的使用期限为 7 天。7 天后，您可以重复使用它。如果您在令牌使用期限内重复使用令牌，则会出现以下情况：

- 如果您通过相同的 Start 操作和相同的输入参数重复使用此令牌，则将返回相同的 JobId。此任务不会再次执行，Amazon Rekognition Video 不会向注册的 Amazon SNS 主题发送完成状态。
- 如果您对相同的 Start 操作重复使用此令牌，并且只进行了细微的输入参数更改，则会引发 IdempotentParameterMismatchException（HTTP 状态代码：400）异常。
- 您不应该通过其他 Start 操作重复使用令牌，因为从 Amazon Rekognition 会得到不可预测的结果。

对 StartLabelDetection 操作的响应是作业标识符 (JobId)。使用 JobId 可跟踪请求并在 Amazon Rekognition Video 将完成状态发布到 Amazon SNS 主题之后获取分析结果。例如：

```
{"JobId":"270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3"}
```

如果您以并发方式启动的作业过多，则调用 StartLabelDetection 会引发 LimitExceededException（HTTP 状态代码：400），直到并发运行的作业数量低于 Amazon Rekognition 服务限制。

如果您发现活动量陡增时会引发 LimitExceededException 异常，请考虑使用 Amazon SQS 队列管理传入请求。如果您发现 Amazon SQS 队列无法管理您的平均并发请求数，并且仍然收到LimitExceededException异常，请联系 AWS 支持人员。

获取 Amazon Rekognition Video 分析请求的完成状态

Amazon Rekognition Video 会将分析完成通知发送到注册的 Amazon SNS 主题。通知将在 JSON 字符串中包含操作的任务标识符和完成状态。成功的视频分析请求具有 SUCCEEDED 状态。例如，以下结果展示了标签检测任务的成功处理。

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnn",
    "Status": "SUCCEEDED",
    "API": "StartLabelDetection",
    "JobTag": "DetectingLabels",
    "Timestamp": 1510865364756,
    "Video": {
        "S3ObjectName": "video.mp4",
        "S3Bucket": "amzn-s3-demo-bucket"
    }
}
```

有关更多信息，请参阅 参考：视频分析结果通知。

要获取由 Amazon Rekognition Video 发布到 Amazon SNS 主题的状态信息，请使用以下选项之一：

- AWS Lambda – 您可订阅写入到 Amazon SNS 主题的 AWS Lambda 函数。此函数在 Amazon Rekognition 通知 Amazon SNS 主题请求已完成时调用。如果您希望服务器端代码处理视频分析请求的结果，请使用 Lambda 函数。例如，在将信息返回到客户端应用程序之前，您可能希望使用服务器端代码来注释视频或创建有关视频内容的报告。我们还建议对大型视频进行服务器端处理，因为 Amazon Rekognition API 可能返回大量数据。
- Amazon Simple Queue Service – 您可以为 Amazon SQS 队列订阅 Amazon SNS 主题。您随后将轮询 Amazon SQS 队列以检索 Amazon Rekognition 在视频分析请求完成后发布的完成状态。有关

更多信息，请参阅 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。如果您希望仅从客户端应用程序调用 Amazon Rekognition Video 操作，请使用 Amazon SQS 队列。

> **Important**
>
> 我们建议不要通过反复调用 Amazon Rekognition Video `Get` 操作来获取请求完成状态。这是因为 Amazon Rekognition Video 会在发出的请求过多时限制 `Get` 操作。如果您同时处理多个视频，那么监控一个 SQS 队列中的完成通知比为每个视频的状态分别轮询 Amazon Rekognition Video 更加简单有效。

获取 Amazon Rekognition Video 分析结果

要获取视频分析请求的结果，请先确保从 Amazon SNS 主题检索的完成状态为 `SUCCEEDED`。然后调用 `GetLabelDetection`，它将传递从 `StartLabelDetection` 返回的 `JobId` 值。请求 JSON 类似于以下示例：

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "SortBy": "TIMESTAMP"
}
```

JobId 是视频分析操作的标识符。由于视频分析会生成大量数据，请使用 `MaxResults` 指定要在单个 Get 操作中返回的结果的最大数量。`MaxResults` 的默认值为 1000。如果您指定的值大于 1000，则返回最多 1000 个结果。如果该操作未返回整个结果集，下一页的分页令牌将在操作响应中返回。如果您来自上一个 Get 请求的分页令牌，请将它与 `NextToken` 结合使用以获取下一页结果。

> **Note**
>
> Amazon Rekognition 将保留视频分析操作的结果 7 天。此时间过后，您将无法检索分析结果。

`GetLabelDetection` 操作响应 JSON 与以下内容类似：

```
{
    "Labels": [
        {
```

```
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Electronics"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.11109819263219833,
                            "Top": 0.08098889887332916,
                            "Left": 0.8881205320358276,
                            "Height": 0.9073750972747803
                        },
                        "Confidence": 99.5831298828125
                    },
                    {
                        "BoundingBox": {
                            "Width": 0.1268676072359085,
                            "Top": 0.14018426835536957,
                            "Left": 0.0003282368124928324,
                            "Height": 0.7993982434272766
                        },
                        "Confidence": 99.46029663085938
                    }
                ],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Person"
            }
```

```
        },
        .
        .
        .

        {
            "Timestamp": 166,
            "Label": {
                "Instances": [],
                "Confidence": 73.6471176147461,
                "Parents": [
                    {
                        "Name": "Clothing"
                    }
                ],
                "Name": "Sleeve"
            }
        }

    ],
    "LabelModelVersion": "2.0",
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Format": "QuickTime / MOV",
        "FrameRate": 23.976024627685547,
        "Codec": "h264",
        "DurationMillis": 5005,
        "FrameHeight": 674,
        "FrameWidth": 1280
    }
}
```

GetLabelDetection 和 GetContentModeration 操作允许您按时间戳或标签名称对分析结果进行分类。您还可以按视频片段或时间戳汇总结果。

您可按检测时间 (视频开始的时间 (毫秒)) 或检测到的实体 (物体、人脸、名人、审阅标签或人员) 的字母顺序为结果排序。要按时间排序，请将 SortBy 输入参数的值设置为 TIMESTAMP。如果未指定 SortBy，则默认行为是按时间排序。上述示例是按时间排序的。要按实体排序，请将 SortBy 输入参数与适用于您执行的操作的值结合使用。例如，要按在对 GetLabelDetection 的调用中检测到的标签排序，请使用值 NAME。

要按时间戳汇总结果，请将 AggregateBy 参数的值设置为 TIMESTAMPS。要按视频片段进行汇总，请将 AggregateBy 的值设置为 SEGMENTS。SEGMENTS 汇总模式将随着时间的推移汇总标签，同

时TIMESTAMPS给出检测到标签的时间戳，使用 2 FPS 采样和每帧输出（注意：当前采样率可能会发生变化，不应假设当前的采样率）。如果未指定值，则默认汇总方法为 TIMESTAMPS。

配置 Amazon Rekognition Video

要将 Amazon Rekognition Video API 用于所存储视频，您必须配置用户和 IAM 服务角色以访问您的 Amazon SNS 主题。您还必须为您的 Amazon SNS 主题订阅 Amazon SQS 队列。

> **Note**
>
> 如果使用这些说明来设置 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 示例，则无需执行步骤 3、4、5 和 6。此示例包括用于创建和配置 Amazon SNS 主题和 Amazon SQS 队列的代码。

本节中的示例通过使用向 Amazon Rekognition Video 提供对多个主题的访问权限的指令，创建新的 Amazon SNS 主题。如果您想要使用现有的 Amazon SNS 主题，请对步骤 3 使用授予访问现有 Amazon SNS 主题的权限。

配置 Amazon Rekognition Video

1. 设置 AWS 账户以访问亚马逊 Rekognition Video。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
2. 安装和配置所需的 AWS SDK。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
3. 要运行本开发人员指南中的代码示例，请确保您选择的用户具有编程访问权限。请参阅授予编程式访问权限了解更多信息。

   您的用户还需要至少以下权限：

   - 亚马逊SQSFull访问权限
   - AmazonRekognitionFullAccess
   - 亚马逊 3 FullAccess
   - 亚马逊SNSFull访问权限

   如果您使用 IAM Identity Center 进行身份验证，请将权限添加到角色的权限集中，否则将权限添加到您的 IAM 角色中。

4. 通过使用 Amazon SNS 控制台创建 Amazon SNS 主题。在主题名称前面加上。AmazonRekognition记下主题的 Amazon 资源名称 (ARN)。确保该主题与您使用的 AWS 端点位于同一区域。
5. 使用 Amazon SQS 控制台创建 Amazon SQS 标准队列。记录队列 ARN。
6. 为队列订阅主题（您在步骤 3 中创建）。
7. 为向 Amazon SQS 队列发送消息的 Amazon SNS 主题授予权限。
8. 创建 IAM 服务角色来为 Amazon Rekognition Video 提供对 Amazon SNS 主题的访问权限。记下服务角色的 Amazon 资源名称 (ARN)。有关更多信息，请参阅 提供对多个 Amazon SNS 主题的访问权限。
9. 为确保您的账户安全，您需要将 Rekognition 的访问范围限制为仅限于您正在使用的资源。这可以通过将信任策略附加到您的 IAM 服务角色来完成。有关如何执行此操作的信息，请参阅 防止跨服务混淆代理。
10. 向您在步骤 1 中创建的用户添加以下内联策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MySid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:Service role ARN from step 7"
        }
    ]
}
```

为该内联策略提供您选择的名称。

11. 如果您使用客户托管 AWS Key Management Service 密钥对 Amazon S3 存储桶中的视频进行加密，请向密钥添加权限，允许您在步骤 7 中创建的服务角色解密视频。服务角色至少需要权限才能执行 kms:GenerateDataKey 和 kms:Decrypt 操作。例如：

```
{
    "Sid": "Decrypt only",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/user from step 1"
    },
```

```
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

有关更多信息，请参阅我的 Amazon S3 存储桶使用自定义 AWS KMS 密钥进行默认加密。如何允许用户从存储桶下载并上传到存储桶？以及使用服务器端加密和存储在 AWS Key Management Service 中的 KMS 密钥 (SSE-KMS) 保护数据。

12. 现在您就可以运行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)和使用分析视频 AWS Command Line Interface中的示例了。

提供对多个 Amazon SNS 主题的访问权限

您可使用 IAM 服务角色为 Amazon Rekognition Video 提供对您创建的 Amazon SNS 主题的访问权限。IAM 提供了用于创建 Amazon Rekognition Video 服务角色的 Rekognition 使用案例。

您可以使用权限策略并在主题名称前加上——例如，，来授予亚马逊 Rekognition Video 访问多个亚马逊 SNS 主题`AmazonRekognitionServiceRole`的权限。`AmazonRekognitionAmazonRekognitionMyTopicName`

让 Amazon Rekognition Video 访问多个 Amazon SNS 主题

1. 创建 IAM 服务角色。使用以下信息创建 IAM 服务角色：
    1. 对于服务名称，选择 Rekognition。
    2. 对于服务角色使用案例，选择 Rekognition。您应该会看到列出的`AmazonRekognitionServiceRole`权限策略。 `AmazonRekognitionServiceRole`允许亚马逊 Rekognition Video 访问前缀为的亚马逊 SNS 主题。`AmazonRekognition`
    3. 为该服务角色指定您选择的名称。
2. 记下服务角色的 ARN。您需要它才能开始视频分析操作。

授予访问现有 Amazon SNS 主题的权限

您可以创建一个权限策略，允许 Amazon Rekognition Video 访问现有的 Amazon SNS 主题。

让 Amazon Rekognition Video 访问现有 Amazon SNS 话题

1. 使用 IAM JSON 策略编辑器创建新的权限策略，然后使用以下策略。将 topicarn 替换为所需 Amazon SNS 主题的 Amazon 资源名称 (ARN)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "topicarn"
        }
    ]
}
```

2. 创建 IAM 服务角色，或者更新现有 IAM 服务角色。使用以下信息创建 IAM 服务角色：

   1. 对于服务名称，选择 Rekognition。
   2. 对于服务角色使用案例，选择 Rekognition。
   3. 附加您在步骤 1 中创建的权限策略。

3. 记下服务角色的 ARN。您需要它才能开始视频分析操作。

使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)

本过程展示如何使用 Amazon Rekognition Video 标签检测操作、存储在 Amazon S3 存储桶内的视频和 Amazon SNS 主题来检测视频中的标签。本过程还说明如何使用 Amazon SQS 队列从 Amazon SNS 主题获取完成状态。有关更多信息，请参阅 调用 Amazon Rekognition Video 操作。您不必局限于使用 Amazon SQS 队列。例如，您可以使用 AWS Lambda 函数来获取完成状态。有关更多信息，请参阅使用 Amazon SNS 通知调用 Lambda 函数。

本过程中的示例代码展示如何执行以下操作：

1. 创建 Amazon SNS 主题。
2. 创建 Amazon SQS 队列。
3. 为 Amazon Rekognition Video 提供将视频分析操作的完成状态发布到 Amazon SNS 主题的权限。
4. 为 Amazon SQS 队列订阅 Amazon SNS 主题。

5. 通过调用 StartLabelDetection 启动视频分析请求。
6. 从 Amazon SQS 队列获取完成状态。示例将跟踪 `StartLabelDetection` 中返回的任务标识符 (`JobId`) 并且仅获取与从完成状态读取的任务标识符匹配的结果。如果其他应用程序使用的是同一队列和主题，那么这是一个重要的考量。为简便起见，该示例会删除不匹配的任务。请考虑将它们添加到 Amazon SQS 死信队列以进行进一步调查。
7. 通过调用 GetLabelDetection 来获取和显示视频分析结果。

先决条件

Java 和 Python 提供了此过程的示例代码。您需要安装相应的 AWS SDK。有关更多信息，请参阅 Amazon Rekognition 入门。您使用的 AWS 账户必须具有对 Amazon Rekognition API 的访问权限。有关更多信息，请参阅 Amazon Rekognition 定义的操作。

检测视频中的标签

1. 配置用户对 Amazon Rekognition Video 的访问权限并配置 Amazon Rekognition Video 对 Amazon SNS 的访问权限。有关更多信息，请参阅 配置 Amazon Rekognition Video。您无需执行步骤 3、4、5 和 6，因为示例代码将创建并配置 Amazon SNS 主题和 Amazon SQS 队列。
2. 将 MOV 或 MPEG-4 格式的视频文件上传到 Amazon S3 存储桶。对于测试，请上传时长不超过 30 秒的视频。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下代码示例检测视频中的标签。

   Java

   在函数 `main` 中：

   - 将 `roleArn` 替换为您在配置 Amazon Rekognition Video的步骤 7 中创建的 IAM 服务角色的 ARN。
   - 将 `amzn-s3-demo-bucket` 和 `video` 的值替换为您在步骤 2 中指定的存储桶和视频文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
import com.amazonaws.auth.policy.Policy;
import com.amazonaws.auth.policy.Condition;
import com.amazonaws.auth.policy.Principal;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.SQSActions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CelebrityDetail;
import com.amazonaws.services.rekognition.model.CelebrityRecognition;
import com.amazonaws.services.rekognition.model.CelebrityRecognitionSortBy;
import com.amazonaws.services.rekognition.model.ContentModerationDetection;
import com.amazonaws.services.rekognition.model.ContentModerationSortBy;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.FaceDetection;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.FaceSearchSortBy;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.GetContentModerationRequest;
import com.amazonaws.services.rekognition.model.GetContentModerationResult;
import com.amazonaws.services.rekognition.model.GetFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.GetFaceDetectionResult;
import com.amazonaws.services.rekognition.model.GetFaceSearchRequest;
import com.amazonaws.services.rekognition.model.GetFaceSearchResult;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.GetPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.GetPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Instance;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.NotificationChannel;
import com.amazonaws.services.rekognition.model.Parent;
import com.amazonaws.services.rekognition.model.PersonDetection;
import com.amazonaws.services.rekognition.model.PersonMatch;
import com.amazonaws.services.rekognition.model.PersonTrackingSortBy;
import com.amazonaws.services.rekognition.model.S3Object;
```

```
import
 com.amazonaws.services.rekognition.model.StartCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.StartCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.StartContentModerationRequest;
import com.amazonaws.services.rekognition.model.StartContentModerationResult;
import com.amazonaws.services.rekognition.model.StartFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.StartFaceDetectionResult;
import com.amazonaws.services.rekognition.model.StartFaceSearchRequest;
import com.amazonaws.services.rekognition.model.StartFaceSearchResult;
import com.amazonaws.services.rekognition.model.StartLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.StartLabelDetectionResult;
import com.amazonaws.services.rekognition.model.StartPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.StartPersonTrackingResult;
import com.amazonaws.services.rekognition.model.Video;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.*;

public class VideoDetect {


    private static String sqsQueueName=null;
    private static String snsTopicName=null;
    private static String snsTopicArn = null;
    private static String roleArn= null;
    private static String sqsQueueUrl = null;
    private static String sqsQueueArn = null;
    private static String startJobId = null;
    private static String bucket = null;
    private static String video = null;
    private static AmazonSQS sqs=null;
    private static AmazonSNS sns=null;
    private static AmazonRekognition rek = null;
```

```
    private static NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);


    public static void main(String[] args) throws Exception {

        video = "";
        bucket = "";
        roleArn= "";

        sns = AmazonSNSClientBuilder.defaultClient();
        sqs= AmazonSQSClientBuilder.defaultClient();
        rek = AmazonRekognitionClientBuilder.defaultClient();

        CreateTopicandQueue();

        //=================================================

        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();

       //=================================================


        DeleteTopicandQueue();
        System.out.println("Done!");

    }


    static boolean GetSQSMessageSuccess() throws Exception
    {
     boolean success=false;


        System.out.println("Waiting for job: " + startJobId);
        //Poll queue for messages
        List<Message> messages=null;
        int dotLine=0;
        boolean jobFound=false;
```

```
       //loop until the job status is published. Ignore other messages in
queue.
       do{
           messages = sqs.receiveMessage(sqsQueueUrl).getMessages();
           if (dotLine++<40){
               System.out.print(".");
           }else{
               System.out.println();
               dotLine=0;
           }

           if (!messages.isEmpty()) {
               //Loop through messages received.
               for (Message message: messages) {
                   String notification = message.getBody();

                   // Get status and job id from notification.
                   ObjectMapper mapper = new ObjectMapper();
                   JsonNode jsonMessageTree = mapper.readTree(notification);
                   JsonNode messageBodyText = jsonMessageTree.get("Message");
                   ObjectMapper operationResultMapper = new ObjectMapper();
                   JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                   JsonNode operationJobId = jsonResultTree.get("JobId");
                   JsonNode operationStatus = jsonResultTree.get("Status");
                   System.out.println("Job found was " + operationJobId);
                   // Found job. Get the results and display.
                   if(operationJobId.asText().equals(startJobId)){
                       jobFound=true;
                       System.out.println("Job id: " + operationJobId );
                       System.out.println("Status : " +
operationStatus.toString());
                       if (operationStatus.asText().equals("SUCCEEDED")){
                        success=true;
                       }
                       else{
                           System.out.println("Video analysis failed");
                       }


sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                   }
```

```
                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        //Delete unknown message. Consider moving message to
dead letter queue

sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                    }
                }
            }
            else {
                Thread.sleep(5000);
            }
        } while (!jobFound);

        System.out.println("Finished processing video");
        return success;
    }


    private static void StartLabelDetection(String bucket, String video) throws
Exception{

        NotificationChannel channel= new NotificationChannel()
                .withSNSTopicArn(snsTopicArn)
                .withRoleArn(roleArn);


        StartLabelDetectionRequest req = new StartLabelDetectionRequest()
                .withVideo(new Video()
                        .withS3Object(new S3Object()
                                .withBucket(bucket)
                                .withName(video)))
                .withMinConfidence(50F)
                .withJobTag("DetectingLabels")
                .withNotificationChannel(channel);

        StartLabelDetectionResult startLabelDetectionResult =
rek.startLabelDetection(req);
        startJobId=startLabelDetectionResult.getJobId();

    }

    private static void GetLabelDetectionResults() throws Exception{
```

```
        int maxResults=10;
        String paginationToken=null;
        GetLabelDetectionResult labelDetectionResult=null;

        do {
            if (labelDetectionResult !=null){
                paginationToken = labelDetectionResult.getNextToken();
            }

            GetLabelDetectionRequest labelDetectionRequest= new
GetLabelDetectionRequest()
                    .withJobId(startJobId)
                    .withSortBy(LabelDetectionSortBy.TIMESTAMP)
                    .withMaxResults(maxResults)
                    .withNextToken(paginationToken);


            labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

            VideoMetadata videoMetaData=labelDetectionResult.getVideoMetadata();

            System.out.println("Format: " + videoMetaData.getFormat());
            System.out.println("Codec: " + videoMetaData.getCodec());
            System.out.println("Duration: " +
videoMetaData.getDurationMillis());
            System.out.println("FrameRate: " + videoMetaData.getFrameRate());


            //Show labels, confidence and detection times
            List<LabelDetection> detectedLabels=
labelDetectionResult.getLabels();

            for (LabelDetection detectedLabel: detectedLabels) {
                long seconds=detectedLabel.getTimestamp();
                Label label=detectedLabel.getLabel();
                System.out.println("Millisecond: " + Long.toString(seconds) + "
");

                System.out.println("   Label:" + label.getName());
                System.out.println("   Confidence:" +
detectedLabel.getLabel().getConfidence().toString());

                List<Instance> instances = label.getInstances();
```

```
                System.out.println("   Instances of " + label.getName());
                if (instances.isEmpty()) {
                    System.out.println("        " + "None");
                } else {
                    for (Instance instance : instances) {
                        System.out.println("        Confidence: " +
instance.getConfidence().toString());
                        System.out.println("        Bounding box: " +
instance.getBoundingBox().toString());
                    }
                }
                System.out.println("   Parent labels for " + label.getName() +
":");
                List<Parent> parents = label.getParents();
                if (parents.isEmpty()) {
                    System.out.println("        None");
                } else {
                    for (Parent parent : parents) {
                        System.out.println("        " + parent.getName());
                    }
                }
                System.out.println();
            }
        } while (labelDetectionResult !=null &&
labelDetectionResult.getNextToken() != null);

    }

    // Creates an SNS topic and SQS queue. The queue is subscribed to the
topic.
    static void CreateTopicandQueue()
    {
        //create a new SNS topic
        snsTopicName="AmazonRekognitionTopic" +
Long.toString(System.currentTimeMillis());
        CreateTopicRequest createTopicRequest = new
CreateTopicRequest(snsTopicName);
        CreateTopicResult createTopicResult =
sns.createTopic(createTopicRequest);
        snsTopicArn=createTopicResult.getTopicArn();

        //Create a new SQS Queue
        sqsQueueName="AmazonRekognitionQueue" +
Long.toString(System.currentTimeMillis());
```

```
        final CreateQueueRequest createQueueRequest = new
CreateQueueRequest(sqsQueueName);
        sqsQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
        sqsQueueArn = sqs.getQueueAttributes(sqsQueueUrl,
Arrays.asList("QueueArn")).getAttributes().get("QueueArn");

        //Subscribe SQS queue to SNS topic
        String sqsSubscriptionArn = sns.subscribe(snsTopicArn, "sqs",
sqsQueueArn).getSubscriptionArn();

        // Authorize queue
          Policy policy = new Policy().withStatements(
                  new Statement(Effect.Allow)
                  .withPrincipals(Principal.AllUsers)
                  .withActions(SQSActions.SendMessage)
                  .withResources(new Resource(sqsQueueArn))
                  .withConditions(new
Condition().withType("ArnEquals").withConditionKey("aws:SourceArn").withValues(snsTopic
                  );


          Map queueAttributes = new HashMap();
          queueAttributes.put(QueueAttributeName.Policy.toString(),
policy.toJson());
          sqs.setQueueAttributes(new SetQueueAttributesRequest(sqsQueueUrl,
queueAttributes));


         System.out.println("Topic arn: " + snsTopicArn);
         System.out.println("Queue arn: " + sqsQueueArn);
         System.out.println("Queue url: " + sqsQueueUrl);
         System.out.println("Queue sub arn: " + sqsSubscriptionArn );
     }
    static void DeleteTopicandQueue()
    {
        if (sqs !=null) {
            sqs.deleteQueue(sqsQueueUrl);
            System.out.println("SQS queue deleted");
        }

        if (sns!=null) {
            sns.deleteTopic(snsTopicArn);
            System.out.println("SNS topic deleted");
        }
```

```
    }
}
```

Python

在函数 main 中：

- 将 roleArn 替换为您在配置 Amazon Rekognition Video的步骤 7 中创建的 IAM 服务角色的 ARN。
- 将 amzn-s3-demo-bucket 和 video 的值替换为您在步骤 2 中指定的存储桶和视频文件名。
- 将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。
- 您还可以在设置参数中包含过滤条件。例如，可以在所需值列表旁边使用 LabelsInclusionFilter 或 LabelsExclusionFilter。在下面的代码中，您可以取消对Features和Settings部分的注释，并提供自己的值，将返回的结果限制在您感兴趣的标签上。
- 在对 GetLabelDetection 的调用中，您可以为 SortBy 和 AggregateBy 参数提供值。要按时间排序，请将 SortBy 输入参数的值设置为 TIMESTAMP。要按实体排序，请将 SortBy 输入参数与适用于您执行的操作的值结合使用。要按时间戳汇总结果，请将 AggregateBy 参数的值设置为 TIMESTAMPS。要按视频片段进行汇总，请使用SEGMENTS。

```
## Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import json
import sys
import time

class VideoDetect:

    jobId = ''

    roleArn = ''
    bucket = ''
    video = ''
    startJobId = ''
```

```
    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def __init__(self, role, bucket, video, client, rek, sqs, sns):
        self.roleArn = role
        self.bucket = bucket
        self.video = video
        self.client = client
        self.rek = rek
        self.sqs = sqs
        self.sns = sns

    def GetSQSMessageSuccess(self):

        jobFound = False
        succeeded = False

        dotLine = 0
        while jobFound == False:
            sqsResponse = self.sqs.receive_message(QueueUrl=self.sqsQueueUrl,
MessageAttributeNames=['ALL'],
                                                   MaxNumberOfMessages=10)

            if sqsResponse:

                if 'Messages' not in sqsResponse:
                    if dotLine < 40:
                        print('.', end='')
                        dotLine = dotLine + 1
                    else:
                        print()
                        dotLine = 0
                    sys.stdout.flush()
                    time.sleep(5)
                    continue

                for message in sqsResponse['Messages']:
                    notification = json.loads(message['Body'])
                    rekMessage = json.loads(notification['Message'])
                    print(rekMessage['JobId'])
                    print(rekMessage['Status'])
                    if rekMessage['JobId'] == self.startJobId:
```

```
                        print('Matching Job Found:' + rekMessage['JobId'])
                        jobFound = True
                        if (rekMessage['Status'] == 'SUCCEEDED'):
                            succeeded = True

                        self.sqs.delete_message(QueueUrl=self.sqsQueueUrl,

ReceiptHandle=message['ReceiptHandle'])
                    else:
                        print("Job didn't match:" +
                              str(rekMessage['JobId']) + ' : ' +
self.startJobId)
                    # Delete the unknown message. Consider sending to dead
letter queue
                    self.sqs.delete_message(QueueUrl=self.sqsQueueUrl,

ReceiptHandle=message['ReceiptHandle'])

        return succeeded

    def StartLabelDetection(self):
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
self.bucket, 'Name': self.video}},

NotificationChannel={'RoleArn': self.roleArn,

'SNSTopicArn': self.snsTopicArn},
                                                  MinConfidence=90,
                                                  # Filtration options,
uncomment and add desired labels to filter returned labels
                                                  # Features=['GENERAL_LABELS'],
                                                  # Settings={
                                                  # 'GeneralLabels': {
                                                  # 'LabelInclusionFilters':
['Clothing']
                                                  # }}
                                                   )

        self.startJobId = response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetLabelDetectionResults(self):
        maxResults = 10
        paginationToken = ''
```

```
        finished = False

        while finished == False:
            response = self.rek.get_label_detection(JobId=self.startJobId,
                                                    MaxResults=maxResults,
                                                    NextToken=paginationToken,
                                                    SortBy='TIMESTAMP',
                                                    AggregateBy="TIMESTAMPS")

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for labelDetection in response['Labels']:
                label = labelDetection['Label']

                print("Timestamp: " + str(labelDetection['Timestamp']))
                print("   Label: " + label['Name'])
                print("   Confidence: " + str(label['Confidence']))
                print("   Instances:")
                for instance in label['Instances']:
                    print("      Confidence: " + str(instance['Confidence']))
                    print("      Bounding box")
                    print("        Top: " + str(instance['BoundingBox']['Top']))
                    print("        Left: " + str(instance['BoundingBox']
['Left']))
                    print("        Width: " + str(instance['BoundingBox']
['Width']))
                    print("        Height: " + str(instance['BoundingBox']
['Height']))
                    print()
                print()

                print("Parents:")
                for parent in label['Parents']:
                    print("   " + parent['Name'])

                print("Aliases:")
                for alias in label['Aliases']:
                    print("   " + alias['Name'])
```

```
                print("Categories:")
                for category in label['Categories']:
                    print("   " + category['Name'])
                print("----------")
                print()

                if 'NextToken' in response:
                    paginationToken = response['NextToken']
                else:
                    finished = True

    def CreateTopicandQueue(self):

        millis = str(int(round(time.time() * 1000)))

        # Create SNS topic

        snsTopicName = "AmazonRekognitionExample" + millis

        topicResponse = self.sns.create_topic(Name=snsTopicName)
        self.snsTopicArn = topicResponse['TopicArn']

        # create SQS queue
        sqsQueueName = "AmazonRekognitionQueue" + millis
        self.sqs.create_queue(QueueName=sqsQueueName)
        self.sqsQueueUrl = self.sqs.get_queue_url(QueueName=sqsQueueName)
['QueueUrl']

        attribs = self.sqs.get_queue_attributes(QueueUrl=self.sqsQueueUrl,
                                                AttributeNames=['QueueArn'])
['Attributes']

        sqsQueueArn = attribs['QueueArn']

        # Subscribe SQS queue to SNS topic
        self.sns.subscribe(
            TopicArn=self.snsTopicArn,
            Protocol='sqs',
            Endpoint=sqsQueueArn)

        # Authorize SNS to write SQS queue
        policy = """{{
  "Version":"2012-10-17",
  "Statement":[
```

```
    {{
      "Sid":"MyPolicy",
      "Effect":"Allow",
      "Principal" : {{"AWS" : "*"}},
      "Action":"SQS:SendMessage",
      "Resource": "{}",
      "Condition":{{
        "ArnEquals":{{
          "aws:SourceArn": "{}"
        }}
      }}
    }}
  ]
}}""".format(sqsQueueArn, self.snsTopicArn)

        response = self.sqs.set_queue_attributes(
            QueueUrl=self.sqsQueueUrl,
            Attributes={
                'Policy': policy
            })

    def DeleteTopicandQueue(self):
        self.sqs.delete_queue(QueueUrl=self.sqsQueueUrl)
        self.sns.delete_topic(TopicArn=self.snsTopicArn)

def main():

    roleArn = 'role-arn'
    bucket = 'bucket-name'
    video = 'video-name'

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    rek = boto3.client('rekognition')
    sqs = boto3.client('sqs')
    sns = boto3.client('sns')

    analyzer = VideoDetect(roleArn, bucket, video, client, rek, sqs, sns)
    analyzer.CreateTopicandQueue()

    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess() == True:
        analyzer.GetLabelDetectionResults()
```

```
    analyzer.DeleteTopicandQueue()

if __name__ == "__main__":
    main()
```

Node.Js

请看下面的示例代码：

- 将REGION的值替换为您账户的运营区域名称。
- 将amzn-s3-demo-bucket的值替换为包含您的视频文件的 Amazon S3 存储桶的名称。
- 使用您 Amazon S3 存储桶的视频文件名称替换 videoName 的值。
- 将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。
- 将 roleArn 替换为您在配置 Amazon Rekognition Video的步骤 7 中创建的 IAM 服务角色的 ARN。

```
import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
  SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-
sdk/client-sns";
import  { SQSClient } from "@aws-sdk/client-sqs";
import  { SNSClient } from "@aws-sdk/client-sns";
import  { RekognitionClient, StartLabelDetectionCommand,
 GetLabelDetectionCommand } from "@aws-sdk/client-rekognition";
import { stdout } from "process";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
const profileName = "profile-name"
// Create SNS service object.
const sqsClient = new SQSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const snsClient = new SNSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const rekClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});
```

```
// Set bucket and video variables
const bucket = "bucket-name";
const videoName = "video-name";
const roleArn = "role-arn"
var startJobId = ""

var ts = Date.now();
const snsTopicName = "AmazonRekognitionExample" + ts;
const snsTopicParams = {Name: snsTopicName}
const sqsQueueName = "AmazonRekognitionQueue-" + ts;

// Set the parameters
const sqsParams = {
  QueueName: sqsQueueName, //SQS_QUEUE_URL
  Attributes: {
    DelaySeconds: "60", // Number of seconds delay.
    MessageRetentionPeriod: "86400", // Number of seconds delay.
  },
};

const createTopicandQueue = async () => {
  try {
    // Create SNS topic
    const topicResponse = await snsClient.send(new
 CreateTopicCommand(snsTopicParams));
    const topicArn = topicResponse.TopicArn
    console.log("Success", topicResponse);
    // Create SQS Queue
    const sqsResponse = await sqsClient.send(new
 CreateQueueCommand(sqsParams));
    console.log("Success", sqsResponse);
    const sqsQueueCommand = await sqsClient.send(new
 GetQueueUrlCommand({QueueName: sqsQueueName}))
    const sqsQueueUrl = sqsQueueCommand.QueueUrl
    const attribsResponse = await sqsClient.send(new
 GetQueueAttributesCommand({QueueUrl: sqsQueueUrl, AttributeNames:
 ['QueueArn']}))
    const attribs = attribsResponse.Attributes
    console.log(attribs)
    const queueArn = attribs.QueueArn
    // subscribe SQS queue to SNS topic
    const subscribed = await snsClient.send(new SubscribeCommand({TopicArn:
 topicArn, Protocol:'sqs', Endpoint: queueArn}))
    const policy = {
```

```
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "MyPolicy",
          Effect: "Allow",
          Principal: {AWS: "*"},
          Action: "SQS:SendMessage",
          Resource: queueArn,
          Condition: {
            ArnEquals: {
              'aws:SourceArn': topicArn
            }
          }
        }
      ]
    };

    const response = sqsClient.send(new SetQueueAttributesCommand({QueueUrl:
 sqsQueueUrl, Attributes: {Policy: JSON.stringify(policy)}}))
    console.log(response)
    console.log(sqsQueueUrl, topicArn)
    return [sqsQueueUrl, topicArn]

  } catch (err) {
    console.log("Error", err);
  }
};

const startLabelDetection = async (roleArn, snsTopicArn) => {
  try {
    //Initiate label detection and update value of startJobId with returned Job
 ID
   const labelDetectionResponse = await rekClient.send(new
 StartLabelDetectionCommand({Video:{S3Object:{Bucket:bucket, Name:videoName}},
      NotificationChannel:{RoleArn: roleArn, SNSTopicArn: snsTopicArn}}));
      startJobId = labelDetectionResponse.JobId
      console.log(`JobID: ${startJobId}`)
      return startJobId
  } catch (err) {
    console.log("Error", err);
  }
};

const getLabelDetectionResults = async(startJobId) => {
```

```
 console.log("Retrieving Label Detection results")
 // Set max results, paginationToken and finished will be updated depending on
response values
 var maxResults = 10
 var paginationToken = ''
 var finished = false

 // Begin retrieving label detection results
 while (finished == false){
   var response = await rekClient.send(new GetLabelDetectionCommand({JobId:
startJobId, MaxResults: maxResults,
     NextToken: paginationToken, SortBy:'TIMESTAMP'}))
     // Log metadata
     console.log(`Codec: ${response.VideoMetadata.Codec}`)
     console.log(`Duration: ${response.VideoMetadata.DurationMillis}`)
     console.log(`Format: ${response.VideoMetadata.Format}`)
     console.log(`Frame Rate: ${response.VideoMetadata.FrameRate}`)
     console.log()
     // For every detected label, log label, confidence, bounding box, and
timestamp
     response.Labels.forEach(labelDetection => {
       var label = labelDetection.Label
       console.log(`Timestamp: ${labelDetection.Timestamp}`)
       console.log(`Label: ${label.Name}`)
       console.log(`Confidence: ${label.Confidence}`)
       console.log("Instances:")
       label.Instances.forEach(instance =>{
         console.log(`Confidence: ${instance.Confidence}`)
         console.log("Bounding Box:")
         console.log(`Top: ${instance.Confidence}`)
         console.log(`Left: ${instance.Confidence}`)
         console.log(`Width: ${instance.Confidence}`)
         console.log(`Height: ${instance.Confidence}`)
         console.log()
       })
     console.log()
     // Log parent if found
     console.log("   Parents:")
     label.Parents.forEach(parent =>{
       console.log(`    ${parent.Name}`)
     })
     console.log()
     // Searh for pagination token, if found, set variable to next token
     if (String(response).includes("NextToken")){
```

```
        paginationToken = response.NextToken

      }else{
        finished = true
      }

      })
  }
}

// Checks for status of job completion
const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
  try {
    // Set job found and success status to false initially
    var jobFound = false
    var succeeded = false
    var dotLine = 0
    // while not found, continue to poll for response
    while (jobFound == false){
      var sqsReceivedResponse = await sqsClient.send(new
 ReceiveMessageCommand({QueueUrl:sqsQueueUrl,
        MaxNumberOfMessages:'ALL', MaxNumberOfMessages:10}));
      if (sqsReceivedResponse){
        var responseString = JSON.stringify(sqsReceivedResponse)
        if (!responseString.includes('Body')){
          if (dotLine < 40) {
            console.log('.')
            dotLine = dotLine + 1
          }else {
            console.log('')
            dotLine = 0
          };
          stdout.write('', () => {
            console.log('');
          });
          await new Promise(resolve => setTimeout(resolve, 5000));
          continue
        }
      }

      // Once job found, log Job ID and return true if status is succeeded
      for (var message of sqsReceivedResponse.Messages){
        console.log("Retrieved messages:")
        var notification = JSON.parse(message.Body)
```

```
        var rekMessage = JSON.parse(notification.Message)
        var messageJobId = rekMessage.JobId
        if (String(rekMessage.JobId).includes(String(startJobId))){
          console.log('Matching job found:')
          console.log(rekMessage.JobId)
          jobFound = true
          console.log(rekMessage.Status)
          if (String(rekMessage.Status).includes(String("SUCCEEDED"))){
            succeeded = true
            console.log("Job processing succeeded.")
            var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
          }
        }else{
          console.log("Provided Job ID did not match returned ID.")
          var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
        }
      }
    }
  return succeeded
  } catch(err) {
    console.log("Error", err);
  }
};

// Start label detection job, sent status notification, check for success
 status
// Retrieve results if status is "SUCEEDED", delete notification queue and
 topic
const runLabelDetectionAndGetResults = async () => {
  try {
    const sqsAndTopic = await createTopicandQueue();
    const startLabelDetectionRes = await startLabelDetection(roleArn,
 sqsAndTopic[1]);
    const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startLabelDetectionRes)
    console.log(getSQSMessageSuccess)
    if (getSQSMessageSuccess){
      console.log("Retrieving results:")
      const results = await getLabelDetectionResults(startLabelDetectionRes)
    }
```

```
    const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
 sqsAndTopic[0]}));
    const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
 sqsAndTopic[1]}));
    console.log("Successfully deleted.")
  } catch (err) {
    console.log("Error", err);
  }
};

runLabelDetectionAndGetResults()
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
```

```
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_detect.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class VideoDetect {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <queueUrl> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
         "   video - The name of the video (for example, people.mp4). \n\n" +
         "   queueUrl- The URL of a SQS queue. \n\n" +
         "   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic. \n\n" +
         "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

     if (args.length != 5) {
         System.out.println(usage);
         System.exit(1);
     }

     String bucket = args[0];
     String video = args[1];
     String queueUrl = args[2];
     String topicArn = args[3];
     String roleArn = args[4];
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
```

```
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
            .build();

        SqsClient sqs = SqsClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
            .build();

        NotificationChannel channel = NotificationChannel.builder()
            .snsTopicArn(topicArn)
            .roleArn(roleArn)
            .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs,  queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.recognize_video_detect.main]
    public static void startLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartLabelDetectionRequest labelDetectionRequest =
    StartLabelDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .minConfidence(50F)
                .build();
```

```
        StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

        boolean ans = true;
        String status = "";
        int yy = 0;
        while (ans) {

            GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                .jobId(startJobId)
                .maxResults(10)
                .build();

            GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
            status = result.jobStatusAsString();

            if (status.compareTo("SUCCEEDED") == 0)
                ans = false;
            else
                System.out.println(yy +" status is: "+status);

            Thread.sleep(1000);
            yy++;
        }

        System.out.println(startJobId +" status is: "+status);

    } catch(RekognitionException | InterruptedException e) {
        e.getMessage();
        System.exit(1);
    }
}

public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {

    List<Message> messages;
    ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
        .queueUrl(queueUrl)
        .build();
```

```
    try {
        messages = sqs.receiveMessage(messageRequest).messages();

        if (!messages.isEmpty()) {
            for (Message message: messages) {
                String notification = message.body();

                // Get the status and job id from the notification
                ObjectMapper mapper = new ObjectMapper();
                JsonNode jsonMessageTree = mapper.readTree(notification);
                JsonNode messageBodyText = jsonMessageTree.get("Message");
                ObjectMapper operationResultMapper = new ObjectMapper();
                JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                JsonNode operationJobId = jsonResultTree.get("JobId");
                JsonNode operationStatus = jsonResultTree.get("Status");
                System.out.println("Job found in JSON is " + operationJobId);

                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .build();

                String jobId = operationJobId.textValue();
                if (startJobId.compareTo(jobId)==0) {
                    System.out.println("Job id: " + operationJobId );
                    System.out.println("Status : " +
operationStatus.toString());

                    if (operationStatus.asText().equals("SUCCEEDED"))
                        GetResultsLabels(rekClient);
                    else
                        System.out.println("Video analysis failed");

                    sqs.deleteMessage(deleteMessageRequest);
                }

                else{
                    System.out.println("Job received was not job " +
startJobId);
                    sqs.deleteMessage(deleteMessageRequest);
                }
            }
        }
```

```
    } catch(RekognitionException e) {
        e.getMessage();
        System.exit(1);
    } catch (JsonMappingException e) {
        e.printStackTrace();
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
}

// Gets the job results by calling GetLabelDetection
private static void GetResultsLabels(RekognitionClient rekClient) {

    int maxResults=10;
    String paginationToken=null;
    GetLabelDetectionResponse labelDetectionResult=null;

    try {
        do {
            if (labelDetectionResult !=null)
                paginationToken = labelDetectionResult.nextToken();


            GetLabelDetectionRequest labelDetectionRequest=
GetLabelDetectionRequest.builder()
                .jobId(startJobId)
                .sortBy(LabelDetectionSortBy.TIMESTAMP)
                .maxResults(maxResults)
                .nextToken(paginationToken)
                .build();

            labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
            VideoMetadata videoMetaData=labelDetectionResult.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());

            List<LabelDetection> detectedLabels= labelDetectionResult.labels();
            for (LabelDetection detectedLabel: detectedLabels) {
                long seconds=detectedLabel.timestamp();
                Label label=detectedLabel.label();
```

```
                System.out.println("Millisecond: " + seconds + " ");

                System.out.println("   Label:" + label.name());
                System.out.println("   Confidence:" +
 detectedLabel.label().confidence().toString());

                List<Instance> instances = label.instances();
                System.out.println("   Instances of " + label.name());

                if (instances.isEmpty()) {
                    System.out.println("        " + "None");
                } else {
                    for (Instance instance : instances) {
                        System.out.println("        Confidence: " +
 instance.confidence().toString());
                        System.out.println("        Bounding box: " +
 instance.boundingBox().toString());
                    }
                }
                System.out.println("   Parent labels for " + label.name() +
 ":");
                List<Parent> parents = label.parents();

                if (parents.isEmpty()) {
                    System.out.println("        None");
                } else {
                    for (Parent parent : parents) {
                        System.out.println("   " + parent.name());
                    }
                }
                System.out.println();
            }
        } while (labelDetectionResult !=null &&
 labelDetectionResult.nextToken() != null);

    } catch(RekognitionException e) {
        e.getMessage();
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_detect.main]
}
```

4. 构建并运行代码。此操作可能需要一段时间才能完成。完成后，将显示在视频中检测到的标签的列表。有关更多信息，请参阅 检测视频中的标签。

使用分析视频 AWS Command Line Interface

你可以使用 AWS Command Line Interface (AWS CLI) 调用 Amazon Rekognition Video 操作。设计模式与在或其他 AWS 上使用亚马逊 Rekognition Video API 相同。 适用于 Java 的 AWS SDK SDKs有关更多信息，请参阅 Amazon Rekognition Video API 概述。以下过程说明如何使用 AWS CLI 来检测视频中的标签。

您可通过调用 `start-label-detection` 开始检测视频中的标签。当 Amazon Rekognition 分析完视频后，完成状态将发送到在 `start-label-detection` 的 `--notification-channel` 参数中指定的 Amazon SNS 主题。您可通过为 Amazon Simple Queue Service (Amazon SQS) 队列订阅 Amazon SNS 主题来获取完成状态。然后轮询 receive-message 以从 Amazon SQS 队列获取完成状态。

调用 `StartLabelDetection` 时，您可以通过向 `LabelsInclusionFilter` 和/或 `LabelsExclusionFilter` 参数提供过滤参数来筛选结果。有关更多信息，请参阅 检测视频中的标签。

完成状态通知是 `receive-message` 响应内的 JSON 结构。您需要从响应中提取 JSON。有关完成状态 JSON 的信息，请参阅参考：视频分析结果通知。如果已完成状态 JSON 的 `Status` 字段的值为 `SUCCEEDED`，您可通过调用 `get-label-detection` 来获取视频分析请求的结果。调用 `GetLabelDetection` 时，您可以使用 `SortBy` 和 `AggregateBy` 参数对返回的结果进行排序和汇总。

以下过程不包含用于轮询 Amazon SQS 队列的代码。此外，它们也不包含用于分析从 Amazon SQS 队列返回的 JSON 的代码。有关 Java 示例，请参阅使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。

先决条件

要运行此过程，你需要 AWS CLI 安装。有关更多信息，请参阅 Amazon Rekognition 入门。您使用的 AWS 账户必须具有对 Amazon Rekognition API 的访问权限。有关更多信息，请参阅 Amazon Rekognition 定义的操作。

配置 Amazon Rekognition Video 并上传视频

1. 配置用户对 Amazon Rekognition Video 的访问权限并配置 Amazon Rekognition Video 对 Amazon SNS 的访问权限。有关更多信息，请参阅 配置 Amazon Rekognition Video。

2. 将 MOV 或 MPEG-4 格式的视频文件上传到您的 S3 存储桶。在开发和测试时，我们建议使用时长不超过 30 秒的小视频。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

检测视频中的标签

1. 运行以下 AWS CLI 命令开始检测视频中的标签。

```
aws rekognition start-label-detection --video '{"S3Object":{"Bucket":"amzn-s3-demo-
bucket","Name":"video-name"}}' \
 --notification-channel '{"SNSTopicArn":"TopicARN","RoleArn":"RoleARN"}' \
--region region-name  \
--features GENERAL_LABELS \
--profile profile-name \
--settings "{"GeneralLabels":{"LabelInclusionFilters":["Car"]}}
```

   更新以下值：

   - 将amzn-s3-demo-bucket和videofile更改为您在步骤 2 中指定的 Amazon S3 存储桶名称和文件名。
   - 将 us-east-1 更改为您使用的 AWS 区域。
   - 将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。
   - 将 TopicARN 更改为您在 配置 Amazon Rekognition Video 的步骤 3 中创建的 Amazon SNS 主题的 ARN。
   - 将 RoleARN 更改为您在 配置 Amazon Rekognition Video 的步骤 7 中创建的 IAM 服务角色的 ARN。
   - 如果需要，您可以指定 endpoint-url。AWS CLI 应根据提供的区域自动确定正确的端点 URL。但是，如果您使用的是私有 VPC 中的端点，则可能需要指定endpoint-url。AWS 服务端点资源列出了用于指定端点网址的语法以及每个区域的名称和代码。
   - 您还可以在设置参数中包含过滤条件。例如，可以在所需值列表旁边使用 LabelsInclusionFilter 或 LabelsExclusionFilter。

   如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。有关示例，请参阅以下内容：

```
aws rekognition start-label-detection --video "{\"S3Object\":{\"Bucket\":\"amzn-s3-
demo-bucket",\"Name\":\"video-name\"}}" --notification-channel "{\"SNSTopicArn\":
\"TopicARN\",\"RoleArn\":\"RoleARN\"}" \
--region us-east-1 --features GENERAL_LABELS --settings "{\"GeneralLabels\":
{\"LabelInclusionFilters\":[\"Car\"]}}" --profile profile-name
```

2. 记下响应中 JobId 的值。该响应看上去与以下 JSON 示例类似。

```
{
    "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
}
```

3. 编写代码以轮询完成状态 JSON 的 Amazon SQS 队列（通过使用 receive-message）。
4. 编写代码以从完成状态 JSON 提取 Status 字段。
5. 如果的值Status为SUCCEEDED，则运行以下 AWS CLI 命令以显示标签检测结果。

```
aws rekognition get-label-detection  --job-id JobId \
--region us-east-1 --sort-by TIMESTAMP aggregate-by TIMESTAMPS
```

更新以下值：

- 将 JobId 更改得与您在步骤 2 中记下的任务标识符匹配。
- 将 Endpoint 和 us-east-1 更改为您使用的 AWS 端点和区域。

结果看上去与以下示例 JSON 类似：

```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 99.03720092773438,
                "Name": "Speech"
            }
        },
        {
            "Timestamp": 0,
```

```
            "Label": {
                "Confidence": 71.6698989868164,
                "Name": "Pumpkin"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 71.6698989868164,
                "Name": "Squash"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 71.6698989868164,
                "Name": "Vegetable"
            }
        }, .......
```

参考：视频分析结果通知

Amazon Rekognition 将Amazon Rekognition Video 分析请求的结果（包括完成状态）发布到 Amazon Simple Notification Service (Amazon SNS) 主题。要从 Amazon SNS 主题中获取通知，请使用亚马逊简单队列服务队列或函数。 AWS Lambda 有关更多信息，请参阅 the section called “调用 Amazon Rekognition Video 操作”。有关示例，请参阅使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。

负载采用以下 JSON 格式：

```
{
  "JobId": "String",
  "Status": "String",
  "API": "String",
  "JobTag": "String",
  "Timestamp": Number,
  "Video": {
    "S3ObjectName": "String",
    "S3Bucket": "String"
  }
}
```

名称	描述
JobId	任务标识符。匹配从 `Start` 操作返回的任务标识符，如 StartPersonTracking。
状态	任务的状态。有效值为 SUCCEEDED、FAILED 或 ERROR。
API	用于分析输入视频的 Amazon Rekognition Video 操作。
JobTag	任务的标识符。您在对 Start 操作的调用中指定 `JobTag`，如 StartLabelDetection。
Timestamp	任务完成的 Unix 时间戳。
视频	有关已处理的视频的详细信息。包含文件名和将文件存储到的 Amazon S3 存储桶。

以下是已发送到 Amazon SNS 主题的成功通知的示例。

```
{
  "JobId": "6de014b0-2121-4bf0-9e31-856a18719e22",
  "Status": "SUCCEEDED",
  "API": "LABEL_DETECTION",
  "Message": "",
  "Timestamp": 1502230160926,
  "Video": {
    "S3ObjectName": "video.mpg",
    "S3Bucket": "amzn-s3-demo-bucket"
  }
}
```

Amazon Rekognition Video 故障排除

以下内容介绍了有关使用 Amazon Rekognition Video 和所存储视频的问题排查信息。

我从未收到发送到 Amazon SNS 主题的完成状态

在视频分析完成时，Amazon Rekognition Video 会将状态信息发布到 Amazon SNS 主题。通常，您通过订阅带 Amazon SQS 队列或 Lambda 函数的主题来获取完成状态消息。要帮助进行调查，请通过电子邮件订阅 Amazon SNS 主题，以便您的电子邮件收件箱接收发送到 Amazon SNS 主题的消息。有关更多信息，请参阅订阅 Amazon SNS 主题。

如果您的应用程序未收到消息，请考虑：

- 验证分析是否已完成。检查 Get 操作响应中的 `JobStatus` 值（例如 `GetLabelDetection`）。如果该值为 `IN_PROGRESS`，则表示分析未完成，并且尚未将完成状态发布到 Amazon SNS 主题。
- 验证您是否有一个 IAM 服务角色向 Amazon Rekognition Video 授予发布到 Amazon SNS 主题的权限。有关更多信息，请参阅 配置 Amazon Rekognition Video。
- 确认您使用的 IAM 服务角色可以使用角色凭证发布到 Amazon SNS 主题，并且您的服务角色的权限已安全地扩展到您使用的资源。执行以下步骤：
  - 获取用户的 Amazon 资源名称 (ARN)：

```
aws sts get-caller-identity --profile RekognitionUser
```

  - 将用户 ARN 添加到角色信任关系中。有关更多信息，请参阅修改角色。以下示例信任策略指定了用户的角色凭证，并将服务角色的权限限制在您正在使用的资源范围内（有关安全限制服务角色权限范围的更多信息，请参阅防止跨服务混淆代理）：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rekognition.amazonaws.com",
        "AWS": "arn:User ARN"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
      "StringEquals": {
                    "aws:SourceAccount": "Account ID"
                },
       "StringLike": {
                    "aws:SourceArn":
 "arn:aws:rekognition:region:111122223333:streamprocessor/*"
```

```
                }
            }
        }
    ]
}
```

- 代入角色：`aws sts assume-role --role-arn arn:Role ARN --role-session-name SessionName --profile RekognitionUser`
- 发布到 Amazon SNS 主题：`aws sns publish --topic-arn arn:Topic ARN --message "Hello World!" --region us-east-1 --profile RekognitionUser`

如果 AWS CLI 命令起作用，则会收到消息（如果您通过电子邮件订阅了主题，则在电子邮件收件箱中）。如果您没有收到消息：

- 检查您是否配置了 Amazon Rekognition Video。有关更多信息，请参阅 配置 Amazon Rekognition Video。
- 查看此排查问题的其他提示。

- 检查您使用了正确的 Amazon SNS 主题：
  - 如果您使用 IAM 服务角色向 Amazon Rekognition Video 提供对单个 Amazon SNS 主题的访问权限，请确保已向您授予对正确的 Amazon SNS 主题的权限。有关更多信息，请参阅 授予访问现有 Amazon SNS 主题的权限。
  - 如果您使用 IAM 服务角色向 Amazon Rekognition Video 授予对多个 SNS 主题的访问权限，请验证您使用的主题是否正确，并且主题名称前面是否带有。AmazonRekognition有关更多信息，请参阅 提供对多个 Amazon SNS 主题的访问权限。
  - 如果您使用 AWS Lambda 函数，请确认您的 Lambda 函数已订阅正确的亚马逊 SNS 主题。有关更多信息，请参阅扇出到 Lambda 函数。
- 如果您使用 Amazon SQS 队列订阅 Amazon SNS 主题，请确认 Amazon SNS 主题有权将消息发送到 Amazon SQS 队列。有关更多信息，请参阅为向 Amazon SQS 队列发送消息的 Amazon SNS 主题授予权限。

我需要更多帮助来排除 Amazon SNS 主题的故障

您可以 AWS X-Ray 与 Amazon SNS 配合使用来跟踪和分析通过您的应用程序传输的消息。有关更多信息，请参阅 Amazon SNS 和。 AWS X-Ray

如需更多帮助，您可以将问题发布到 Amazon Rekognition 论坛 或考虑注册获取 AWS 技术支持。

使用流视频事件

您可以使用 Amazon Rekognition Video 检测并识别流视频中的人脸或检测其中的对象。Amazon Rekognition Video 使用 Amazon Kinesis Video Streams 来接收和处理视频流。您可以创建流处理器，其参数显示希望流处理器从视频流中检测的内容。Rekognition 将来自流视频事件的标签检测结果作为 Amazon SNS 和 Amazon S3 通知发送。Rekognition 将人脸搜索结果输出到 Kinesis 数据流。

人脸搜索流处理器使用 `FaceSearchSettings` 从集合中搜索人脸。有关如何采用人脸搜索流处理器来分析流视频中的人脸的更多信息，请参阅the section called “在流视频中搜索集合中的人脸”。

标签检测流处理器使用 `ConnectedHomeSettings` 搜索流视频事件中的人员、包裹和宠物。有关如何采用标签检测流处理器的更多信息，请参阅the section called “检测流视频事件中的标签”。

Amazon Rekognition Video 流处理器操作概述

您可通过启动 Amazon Rekognition Video 流处理器并将视频流式传输到 Amazon Rekognition Video 中来开始分析流视频。利用 Amazon Rekognition Video 流处理器，您可以启动、停止和管理流处理器。通过调用 CreateStreamProcessor 创建流处理器。创建人脸搜索流处理器的请求参数包括 Kinesis 视频流的 Amazon 资源名称 (ARNs)、Kinesis 数据流以及用于识别流视频中人脸的集合标识符。创建安全监控流处理器的请求参数包括 Kinesis 视频流和 Amazon SNS 主题的 Amazon 资源名称 (ARNs)、您要在视频流中检测到的对象类型以及输出结果的 Amazon S3 存储桶信息。它还包含您为流处理器指定的名称。

您可通过调用 StartStreamProcessor 操作来开始处理视频。要获取流处理器的状态信息，请调用 DescribeStreamProcessor。您可以调用的其他TagResource操作包括标记流处理器和DeleteStreamProcessor删除流处理器。如果您使用的是人脸搜索流处理器，也可以使用StopStreamProcessor停止流处理器。要获取您账户中的流处理器列表，请调用 ListStreamProcessors。

在流处理器开始运行后，可通过您在 `CreateStreamProcessor` 中指定的 Kinesis 视频流将视频流式传输到 Amazon Rekognition Video 中。你可以使用 Kinesis Video Streams PutMediaSDK 操作将视频传送到 Kinesis 视频流中。有关示例，请参阅 PutMediaAPI 示例。

有关您的应用程序如何使用来自人脸搜索流处理器的 Amazon Rekognition Video 分析结果的信息，请参阅读取流视频分析结果。

向新的流处理器添加标签

可以使用标签识别、整理、搜索和筛选 Amazon Rekognition 流处理器。每个标签都是由用户定义的键和值组成的标签。

您可以在使用 CreateStreamProcessor 操作创建流处理器时向其添加标签。在Tags数组输入参数中指定一个或多个标签。以下是带有标签的 CreateStreamProcessor 请求的 JSON 示例。

```
{
       "Name": "streamProcessorForCam",
       "Input": {
              "KinesisVideoStream": {
                     "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/
inputVideo"
              }
       },
       "Output": {
              "KinesisDataStream": {
                     "Arn": "arn:aws:kinesis:us-east-1:nnnnnnnnnnnn:stream/outputData"
              }
       },
       "RoleArn": "arn:aws:iam::nnnnnnnnnnn:role/roleWithKinesisPermission",
       "Settings": {
              "FaceSearch": {
                     "CollectionId": "collection-with-100-faces",
                     "FaceMatchThreshold": 85.5
              },
              "Tags": {
      "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
       }
}
```

向现有流处理器添加标签

可以使用标签识别、整理、搜索和筛选 Amazon Rekognition 流处理器。每个标签都是由用户定义的键和值组成的标签。

要将一个或多个标签添加到现有流处理器，请使用 TagResource 操作。指定流处理器的 Amazon 资源名称（ARN）（ResourceArn）和要添加的标签（Tags）。以下示例说明了如何添加两个标签。

```
aws rekognition tag-resource --resource-arn resource-arn \
               --tags '{"key1":"value1","key2":"value2"}'
```

> **Note**
>
> 如果您不了解流处理器的 Amazon 资源名称，则可以使用 `DescribeStreamProcessor` 操作。

列出流处理器中的标签

可以使用标签识别、整理、搜索和筛选 Amazon Rekognition 流处理器。每个标签都是由用户定义的键和值组成的标签。

要列出附加到流处理器的标签，请使用 `ListTagsForResource` 操作并指定流处理器的 ARN（`ResourceArn`）。响应是附加到指定流处理器的标签键和值的映射。

```
aws rekognition list-tags-for-resource --resource-arn resource-arn
```

输出将显示附加到流处理器的标签列表：

```
{
    "Tags": {
        "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
    }
}
```

删除流处理器中的标签

可以使用标签识别、整理、搜索和筛选 Amazon Rekognition 流处理器。每个标签都是由用户定义的键和值组成的标签。

要从流处理器中删除一个或多个标签，请使用 `UntagResource` 操作。指定模型的 ARN（`ResourceArn`）和要移除的标签键（`Tag-Keys`）。

```
aws rekognition untag-resource --resource-arn resource-arn \
               --tag-keys '["key1","key2"]'
```

或者，您可以按以下格式指定标签键：

```
--tag-keys key1,key2
```

错误处理

本节描述运行时系统错误，以及如何处理它们。它还描述特定于 Amazon Rekognition 的错误消息和代码。

主题

- 错误组成部分
- 错误消息和代码
- 应用程序中的错误处理

错误组成部分

当程序发送一个请求时，Amazon Rekognition 会尝试处理该请求。如果请求成功，Amazon Rekognition 将返回一个 HTTP 成功状态代码（200 OK），以及所请求操作的结果。

如果该请求不成功，Amazon Rekognition 会返回一个错误。每个错误包含三个部分：

- HTTP 状态代码（如 400）。
- 异常名称（如 InvalidS3ObjectException）。
- 错误消息（如 Unable to get object metadata from S3. Check object key, region and/or access permissions.）。

AWS SDKs 负责将错误传播到您的应用程序，以便您可以采取适当的措施。例如，在 Java 程序中，您可以编写 try-catch 逻辑以处理 ResourceNotFoundException。

如果您使用的不是 AWS SDK，将需要分析来自 Amazon Rekognition 的低级响应的内容。以下是此类响应的示例：

```
HTTP/1.1 400 Bad Request
Content-Type: application/x-amz-json-1.1
Date: Sat, 25 May 2019 00:28:25 GMT
x-amzn-RequestId: 03507c9b-7e84-11e9-9ad1-854a4567eb71
Content-Length: 222
```

```
Connection: keep-alive

{"__type":"InvalidS3ObjectException","Code":"InvalidS3ObjectException","Logref":"5022229e-7e48-
 to get object metadata from S3. Check object key, region and/or access permissions."}
```

错误消息和代码

以下是 Amazon Rekognition 返回的异常列表，按 HTTP 状态代码分组。如果确定重试？为是，则可以在此提交相同请求。如果确定重试？为否，则需要先解决客户端问题，然后再提交新请求。

HTTP 状态代码 400

HTTP 400 状态代码表示与请求相关的问题。有些问题示例是关于身份验证失败、缺少必需参数或超出操作的预配置吞吐量。必须先解决应用程序中存在的问题，然后再重新提交请求。

AccessDeniedException

消息：调用操作时出错 (AccessDeniedException)：用户：无权在资源：上执行：。 <Operation><User ARN><Operation><Resource ARN>

您无权执行该操作。使用授权用户或 IAM 角色的 Amazon 资源名称 (ARN) 来执行操作。

确定重试？ 否

GroupFacesInProgressException

消息：无法安排 GroupFaces 作业。此集合已经有 GroupFaces 任务。

在现有作业完成后重试该操作。

确定重试？ 否

IdempotentParameterMismatchException

消息 ClientRequestToken：您提供的:已在使用中。 <Token>

ClientRequestToken 输入参数已在操作中重复使用，但其他输入参数中至少有一个与之前对该操作的调用不同。

确定重试？ 否

ImageTooLargeException

消息：图像大小太大。

输入的图像尺寸超过允许的上限。如果您正在呼叫 DetectProtectiveEquipment，则图像大小或分辨率超过允许的限制。有关更多信息，请参阅 Amazon Rekognition 中的准则和配额。

确定重试？ 否

InvalidImageFormatException

消息：请求具有无效的图像格式。

提供的图像格式不受支持。使用受支持的图像格式（.JPEG 和 .PNG）。有关更多信息，请参阅 Amazon Rekognition 中的准则和配额。

确定重试？ 否

InvalidPaginationTokenException

消息

- 令牌无效
- 分页标记无效

请求中的分页标记无效。标记可能已过期。

确定重试？ 否

InvalidParameterException

消息：请求具有无效参数。

有一个输入参数违反了约束。先验证您的参数，然后重新调用 API 操作。

确定重试？ 否

无效3 ObjectException

消息：

- 请求具有无效的 S3 对象。
- 无法从 S3 获取对象元数据。请检查对象键、区域和/或访问权限。

Amazon Rekognition 无法访问请求中指定的 S3 对象。有关更多信息，请参阅配置对 S3 的访问：AWS S3 管理访问。有关问题排查信息，请参阅 Amazon S3 问题排查。

确定重试？ 否

LimitExceededException

消息：

- 超出账户的流处理器限制，限制 - <当前限制>。
- 用户 <用户 ARN> 具有 <打开的作业数> 个打开的作业，最大限制：<最大限制>

超出了 Amazon Rekognition 服务限制。例如，如果您并发启动了太多 Amazon Rekognition Video 作业，则调用启动操作（例如 `StartLabelDetection`）会引发 `LimitExceededException` 异常（HTTP 状态代码：400），直到并发运行的作业数量低于 Amazon Rekognition 服务限制。

确定重试？ 否

ProvisionedThroughputExceededException

消息：

- 超出了预配置的速率。
- 超出了 S3 下载限制。

请求数超出了您的吞吐量限制。有关更多信息，请参阅 Amazon Rekognition 服务限制。

要请求提高限制，请按照 the section called “创建更改 TPS 配额的案例” 中的说明进行操作。

确定重试？ 是

ResourceAlreadyExistsException

消息：集合 ID <集合 ID> 已存在。

具有指定 ID 的集合已存在。

确定重试？ 否

ResourceInUseException

消息：

- 流处理器名称已在使用。
- 指定的资源正在使用中。
- 处理器不可用于停止流。
- 无法删除流处理器。

在资源可用时重试。

确定重试？ 否

ResourceNotFoundException

消息：各个消息取决于 API 调用。

指定的资源不存在。

确定重试？ 否

ThrottlingException

消息：减慢；请求速率突然增加。

您的请求速率增加过快。减慢您的请求速率并逐渐提高它。我们建议您进行指数回退并重试。默认情况下，AWS SDKs 使用自动重试逻辑和指数退避。有关更多信息，请参阅 AWS 中的错误重试和指数回退和指数回退和抖动。

确定重试？ 是

VideoTooLargeException

消息：视频大小（以字节为单位）<视频大小> 已超出最大限制，即 <最大大小> 字节。

所提供媒体的文件大小或持续时间过大。有关更多信息，请参阅 Amazon Rekognition 中的准则和配额。

确定重试？ 否

HTTP 状态代码 5xx

HTTP 5xx 状态代码表示必须由 AWS解决的问题。This might be a transient error。如果是临时错误，您可以重试请求，直到操作成功。否则，请转至 AWS 服务运行状况控制面板以查看是否存在与服务相关的任何操作问题。

InternalServerError (HTTP 500)

消息：Internal server error (内部服务器错误)

Amazon Rekognition 遇到了一个服务问题。重新尝试您的调用。您应该执行指数级退避并重试。默认情况下， AWS SDKs 使用自动重试逻辑和指数退避。有关更多信息，请参阅 AWS 中的错误重试和指数回退和指数回退和抖动。

确定重试？ 是

ThrottlingException (HTTP 500)

消息：服务不可用

Amazon Rekognition 暂时无法处理该请求。重新尝试您的调用。我们建议您进行指数回退并重试。默认情况下， AWS SDKs 使用自动重试逻辑和指数退避。有关更多信息，请参阅 AWS 中的错误重试和指数回退和指数回退和抖动。

确定重试？ 是

应用程序中的错误处理

为了让应用程序平稳运行，您需要添加逻辑以捕获错误并做出响应。典型的方法包括使用 `try-catch` 块或 `if-then` 语句。

AWS 自行 SDKs 执行重试和错误检查。如果您在使用其中一个 AWS 时遇到错误 SDKs，则错误代码和描述可以帮助您排除故障。

您还应该会在响应中看到 `Request ID`。如果您需要使用 AWS Support 来诊断问题，则 `Request ID` 会很有用。

以下 Java 代码段尝试检测图像中的对象并执行基本错误处理。（在这种情况下，它会向用户告知请求失败。）

```
try {
    DetectLabelsResult result = rekognitionClient.detectLabels(request);
    List <Label> labels = result.getLabels();

    System.out.println("Detected labels for " + photo);
    for (Label label: labels) {
        System.out.println(label.getName() + ": " + label.getConfidence().toString());
    }
}
catch(AmazonRekognitionException e) {
    System.err.println("Could not complete operation");
    System.err.println("Error Message:  " + e.getMessage());
    System.err.println("HTTP Status:    " + e.getStatusCode());
    System.err.println("AWS Error Code: " + e.getErrorCode());
    System.err.println("Error Type:     " + e.getErrorType());
    System.err.println("Request ID:     " + e.getRequestId());
}
catch (AmazonClientException ace) {
    System.err.println("Internal error occurred communicating with Rekognition");
    System.out.println("Error Message:  " + ace.getMessage());
}
```

在此代码段中，`try-catch` 结构处理两种不同的异常：

- `AmazonRekognitionException` – 如果客户端请求正确地传输到 Amazon Rekognition，但 Amazon Rekognition 无法处理请求并返回错误响应，会出现此异常。
- `AmazonClientException` – 如果客户端无法从服务获得响应或者客户端无法解析来自服务的响应，会出现此异常。

将 Amazon Rekognition 用作 FedRAMP 授权服务

AWS FedRAMP 合规计划将 Amazon Rekognition 列为一项经过 FedRAMP 授权的服务。如果您是联邦或商业客户，则可以使用该服务来处理和存储 AWS 美国东部和美国西部地区的敏感工作负载，其数据最高可达中等影响级别。您可以将该服务用于 AWS GovCloud（美国）地区授权边界内的敏感工作负载，数据最高可达高影响级别。有关 FedRAMP 合规性的更多信息，请参阅 AWS FedRAMP 合规性。

要符合 FedRAMP 要求，您可以使用联邦信息处理标准 (FIPS) 端点。这样一来，您在处理敏感信息时便可访问 FIPS 140-2 验证的加密模块。有关 FIPS 端点的更多信息，请参阅 FIPS 140-2 概述。

您可以使用 AWS Command Line Interface (AWS CLI) 或 AWS 中的一个 SDKs 来指定 Amazon Rekognition 使用的终端节点。

有关可用于 Amazon Rekognition 的端点，请参阅 Amazon Rekognition 区域和端点。

以下是《Amazon Rekognition 开发人员指南》中列出集合主题的示例。这些示例进行了修改，以指定区域和通过其访问 Amazon Rekognition 的 FIPS 端点。

Java

对于 Java，在构建 Amazon Rekognition 客户端时可使用 `withEndpointConfiguration` 方法。以下示例显示了在美国东部（弗吉尼亚北部）区域使用 FIPS 端点的集合：

```
//Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;

public class ListCollections {

   public static void main(String[] args) throws Exception {


      AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.standard()
         .withEndpointConfiguration(new
 AwsClientBuilder.EndpointConfiguration("https://rekognition-fips.us-
east-1.amazonaws.com","us-east-1"))
         .build();


      System.out.println("Listing collections");
      int limit = 10;
      ListCollectionsResult listCollectionsResult = null;
      String paginationToken = null;
```

```
      do {
         if (listCollectionsResult != null) {
            paginationToken = listCollectionsResult.getNextToken();
         }
         ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
                 .withMaxResults(limit)
                 .withNextToken(paginationToken);

 listCollectionsResult=amazonRekognition.listCollections(listCollectionsRequest);

         List < String > collectionIds = listCollectionsResult.getCollectionIds();
         for (String resultId: collectionIds) {
            System.out.println(resultId);
         }
      } while (listCollectionsResult != null &&
 listCollectionsResult.getNextToken() !=
         null);

   }
}
```

AWS CLI

对于 AWS CLI，使用`--endpoint-url`参数指定访问亚马逊 Rekognition 的终端节点。以下示例显示了在美国东部（俄亥俄州）区域使用 FIPS 端点的集合：

```
aws rekognition list-collections --endpoint-url https://rekognition-fips.us-
east-2.amazonaws.com --region us-east-2
```

Python

对于 Python，可在 boto3.client 函数中使用 `endpoint_url` 参数。将其设置为要指定的端点。以下示例显示了在美国西部（俄勒冈州）区域使用 FIPS 端点的集合：

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def list_collections():
```

```
    max_results=2

    client=boto3.client('rekognition', endpoint_url='https://rekognition-fips.us-
west-2.amazonaws.com', region_name='us-west-2')

    #Display all the collections
    print('Displaying collections...')
    response=client.list_collections(MaxResults=max_results)
    collection_count=0
    done=False

    while done==False:
        collections=response['CollectionIds']

        for collection in collections:
            print (collection)
            collection_count+=1
        if 'NextToken' in response:
            nextToken=response['NextToken']

 response=client.list_collections(NextToken=nextToken,MaxResults=max_results)

        else:
            done=True

    return collection_count

def main():

    collection_count=list_collections()
    print("collections: " + str(collection_count))
if __name__ == "__main__":
    main()
```

传感器、输入图像和视频的最佳实践

本节包含使用 Amazon Rekognition 的最佳实践信息。这些最佳实践将帮助您从调用的操作中获得出色性能。如果您难以从一项操作中获得预期结果，请确保遵循此处记录的最佳实践。

有关图像操作延迟的信息，请参阅以下内容：

- the section called “Amazon Rekognition Image 操作延迟”

人脸比较和人脸搜索操作要求您遵循特定的最佳实践来查找图像中的人脸。这些要求也记录在以下链接中：

- the section called “有关面部比较输入图像的建议”
- the section called “在集合中搜索人脸的建议”

以下几节介绍如何为 Amazon Rekognition 能够分析的每种媒体类型设置摄像机：

- the section called “针对摄像机设置（图像和视频）的建议”
- the section called “针对摄像机设置（存储视频和流视频）的建议”
- the section called “针对摄像机设置（流视频）的建议”

Face Liveness 操作也有自己的最佳实践，应遵循这些最佳实践，才能从 Liveness 检查工具中获得出色性能：

- the section called “Face Liveness 的使用建议”

Amazon Rekognition Image 操作延迟

要确保 Amazon Rekognition Image 操作的延迟尽可能最低，请考虑以下事项：

- 包含您的图像的 Amazon S3 存储桶的区域必须与您用于 Amazon Rekognition Image API 操作的区域匹配。
- 使用图像字节调用 Amazon Rekognition Image 操作快于将图像上传到 Amazon S3 存储桶之后在 Amazon Rekognition Image 操作中引用上传的图像。如果您将图像上传到 Amazon Rekognition Image 以进行实时处理，请考虑此方法。例如，从 IP 摄像机上传的图像和通过 Web 门户上传的图像。

- 如果图像已位于 Amazon S3 存储桶中，则在 Amazon Rekognition Image 操作中引用它可能快于将图像字节传递到该操作。

有关面部比较输入图像的建议

用于人脸比较操作的模型适用于各种姿势、面部表情、年龄范围、旋转、照明条件和大小。我们建议您在为 CompareFaces 添加参考照片时或使用 IndexFaces 将人脸添加到集合时遵循以下准则。

输入图像进行人脸操作的一般建议

- 使用明亮清晰的图像。尽可能避免使用因主体和摄像机移动而模糊的图像。DetectFaces 可用于确定人脸的亮度和清晰度。
- 为了进行视线检测，建议您以原始大小和质量上传原始图像。
- 使用人脸处于建议的角度范围内的图像。俯仰角在朝下时应小于 30 度，在朝上时应小于 45 度。偏航角在任一方向应小于 45 度。翻滚角没有限制。
- 使用双眼睁开并且可见的人脸的图像。
- 使用人脸未遮盖或紧密裁剪的图像。图像应包含人的完整头部和肩部。它不应该被裁剪到面部边界框。
- 避免挡住人脸的物品（如头巾和口罩）。
- 使用人脸占比较高的图像。人脸占比较高的图像在匹配时的准确度更高。
- 确保图像的分辨率足够大。Amazon Rekognition 可以在高达 1920 x 1080 的图像分辨率下识别小至 50 x 50 像素的人脸。图像的分辨率越高，需要的最小人脸大小越大。大于最小大小的人脸可提供更准确的人脸比较结果集。
- 使用彩色图像。
- 使用人脸上的照明稳定的图像，而不是照明条件不断变化（如阴影）的图像。
- 使用与背景对比鲜明的图像。高对比度单色背景非常适用。
- 对需要高精度的应用使用没有面部表情（嘴巴紧闭且毫无笑容）的人脸的图像。

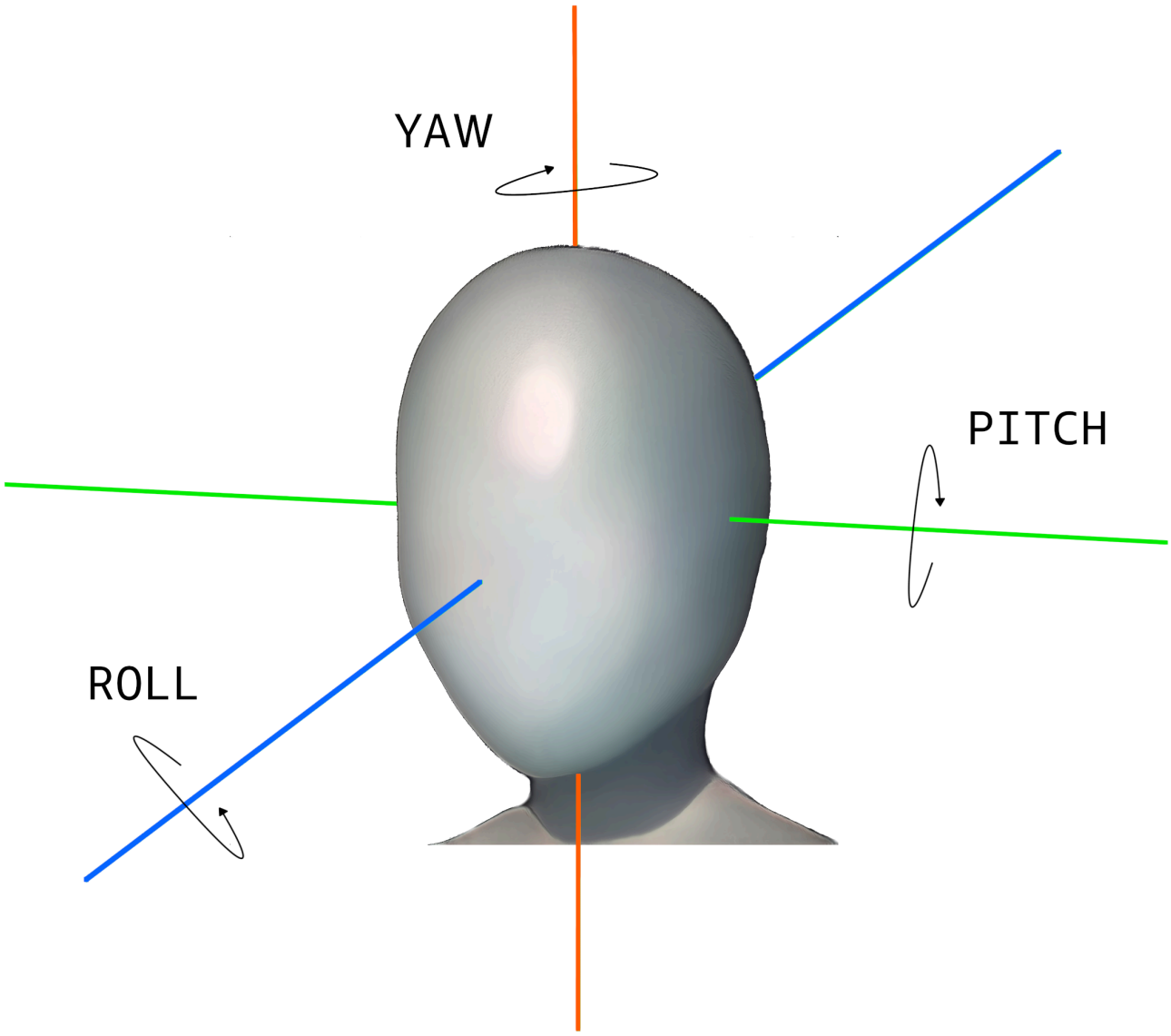
在集合中搜索人脸的建议

- 在集合中搜索人脸时，请确保为最近的人脸图像编制索引。
- 在使用 IndexFaces 创建集合时，使用具有不同的俯仰角和偏航角（位于建议的角度范围内）的个人的多个人脸图像。我们建议至少索引一个人的 5 张图像：直立、脸向左（偏航角为 45 度或更

小）、脸向右（偏航角为 45 度或更小）、脸向下（俯仰角为 30 度或更小）、脸向上（俯仰角为 45 度或更小）。如果您要跟踪属于同一人的这些人脸实例，请考虑使用外部图像 ID 属性，前提是图像中只有一个人脸正在被索引。例如，可以在集合中追踪五张 John Doe 的图像，外部图像 IDs 为`John_Doe_1.jpg, … John_Doe_5.jpg`。

针对摄像机设置（图像和视频）的建议

除了 有关面部比较输入图像的建议 之外，还建议遵循以下建议。



- 图像分辨率 – 图像分辨率没有最低要求，对于总分辨率高达 1920 x 1080 的图像，只要人脸分辨率达到 50 x 50 像素即可。图像的分辨率越高，需要的最小人脸大小越大。

  > **Note**
  >
  > 上述建议基于摄像机的原始分辨率。从低分辨率图像生成高分辨率图像无法产生人脸搜索所需的结果（原因在于图像上采样生成的项目）。

- 摄像机角度 – 摄像机角度有三个测量值 – 俯仰角、翻滚角和偏航角。
  - 俯仰角 – 我们建议在摄像机朝下时采用小于 30 度的俯仰角，在摄像机朝上时采用小于 45 度的俯仰角。
  - 翻滚角 – 此参数没有最低要求。Amazon Rekognition 可以处理任意数量的滚动。
  - 偏航角 – 我们建议偏航角在任一方向都小于 45 度。

  摄像机捕获的沿任何轴的人脸角度是面向场景的摄像机角度与场景中主体头部所处角度的组合。例如，如果摄像机处于朝下 30 度，人的头部处于朝下 30 度，则摄像机拍摄到的实际人脸俯仰角为 60 度。在这种情况下，Amazon Rekognition 将无法识别人脸。我们建议合理地设置摄像机，使它的角度基于以下假设：人们通常以 30 度或更小的整体俯仰角（人脸和摄像机的组合）看着摄像机。
- 摄像机缩放 – 建议的最低人脸分辨率 50 x 50 像素应驱动此摄像机设置。我们建议使用摄影机的缩放设置，使所需人脸的分辨率不低于 50 x 50 像素。
- 摄像机高度 – 建议的摄像机俯仰角应驱动此参数。

针对摄像机设置（存储视频和流视频）的建议

除了 针对摄像机设置（图像和视频）的建议 之外，还建议遵循以下建议。

- 编解码器应为 h.264 编码。
- 建议的帧速率为 30 fps。（它不应小于 5 fps。）
- 建议的编码器比特率为 3 Mbps。（它不应小于 1.5 Mbps。）
- 帧率 vs. 帧分辨率 – 如果编码器比特率是一个约束，我们建议采用更高的帧分辨率而不是更高的帧速率，以获得更好的人脸搜索结果。这可确保 Amazon Rekognition 在分配的比特率内获得最佳帧质量。但是，这也有缺点：由于帧速率低，摄像机将错过场景中的快速运动。务必了解这两个参数针对给定设置的权衡。例如，如果可能的最大比特率为 1.5 Mbps，则摄像机在 5 fps 下可以捕获 1080p，在 15 fps 下可以捕获 720p。选择哪个参数取决于应用，只要达到建议的人脸分辨率 50 x 50 像素即可。

针对摄像机设置（流视频）的建议

除了 针对摄像机设置（存储视频和流视频）的建议 之外，还建议遵循以下建议。

流式处理应用的另一个约束是 Internet 带宽。对于实时视频，Amazon Rekognition 仅接受 Amazon Kinesis Video Streams 作为输入。您应了解编码器比特率和可用网络带宽之间的依赖关系。可用带宽应至少支持摄像机用于对实时流进行编码的相同比特率。这可确保摄像机捕获的任何内容都可通过 Amazon Kinesis Video Streams 进行中继。如果可用带宽低于编码器比特率，Amazon Kinesis Video Streams 基于网络带宽丢弃比特。这会导致视频质量低。

典型流设置涉及将多台摄像机连接到用于中继流的网络中心。在这种情况下，带宽应容纳来自连接到该中心的所有摄像机的流的累积和。例如，如果该中心连接到以 1.5 Mbps 编码的 5 台摄像机，则可用网络带宽至少应为 7.5 Mbps。要确保不会丢失数据包，您应考虑将网络带宽保持在 7.5 Mbps 以上，以应对摄像机和中心之间的连接断开导致的抖动。实际值取决于内部网络的可靠性。

Face Liveness 的使用建议

我们建议您遵循以下最佳实践来使用 Rekognition Face Liveness：

- 用户应在不太暗或太亮且光线相当均匀的环境中完成 Face Liveness 检查。
- 用户在网络浏览器上进行检查时，应将显示屏的亮度调至最大。移动原生设备会自动 SDKs 调整显示屏亮度。
- 选择一个反映您的用例性质的置信度分数阈值。对于具有更高安全性的用例，请使用较高的阈值。
- 定期对审核图像进行人工审查检查，以确保在您设置的置信度阈值下减少仿冒攻击。
- 如果用户对光敏感或不想使用 Rekognition 验证自己的面部活跃度，则为他们提供另一种面部活跃度验证路径。
- 请勿在用户应用程序上发送或显示活跃度检查分数。仅发送成功或失败信号。
- 仅允许在三分钟内对一台设备进行五次失败的活跃度检查。失败五次后，用户将等待 30—60 分钟。如果重复出现该模式 3-5 次，请阻止用户设备进行其他调用。
- 在工作流程中实现准备就绪屏幕，以便用户可以更轻松地通过 Face Liveness 检查。
- 您有责任就 Face Liveness 对内容的处理、存储、使用和传输向您的终端用户提供法律上适当的隐私声明，并征得其必要同意。

检测对象和概念

本节提供了使用 Amazon Rekognition Image 和 Amazon Rekognition Video 检测图像和视频中的标签的信息。

标签是基于其内容在图像或视频中找到的对象或概念（包括场景和动作）。例如，热带海滩上的人物照片可能包含棕榈树（对象）、海滩（场景）、奔跑（动作）以及户外（概念）等标签。

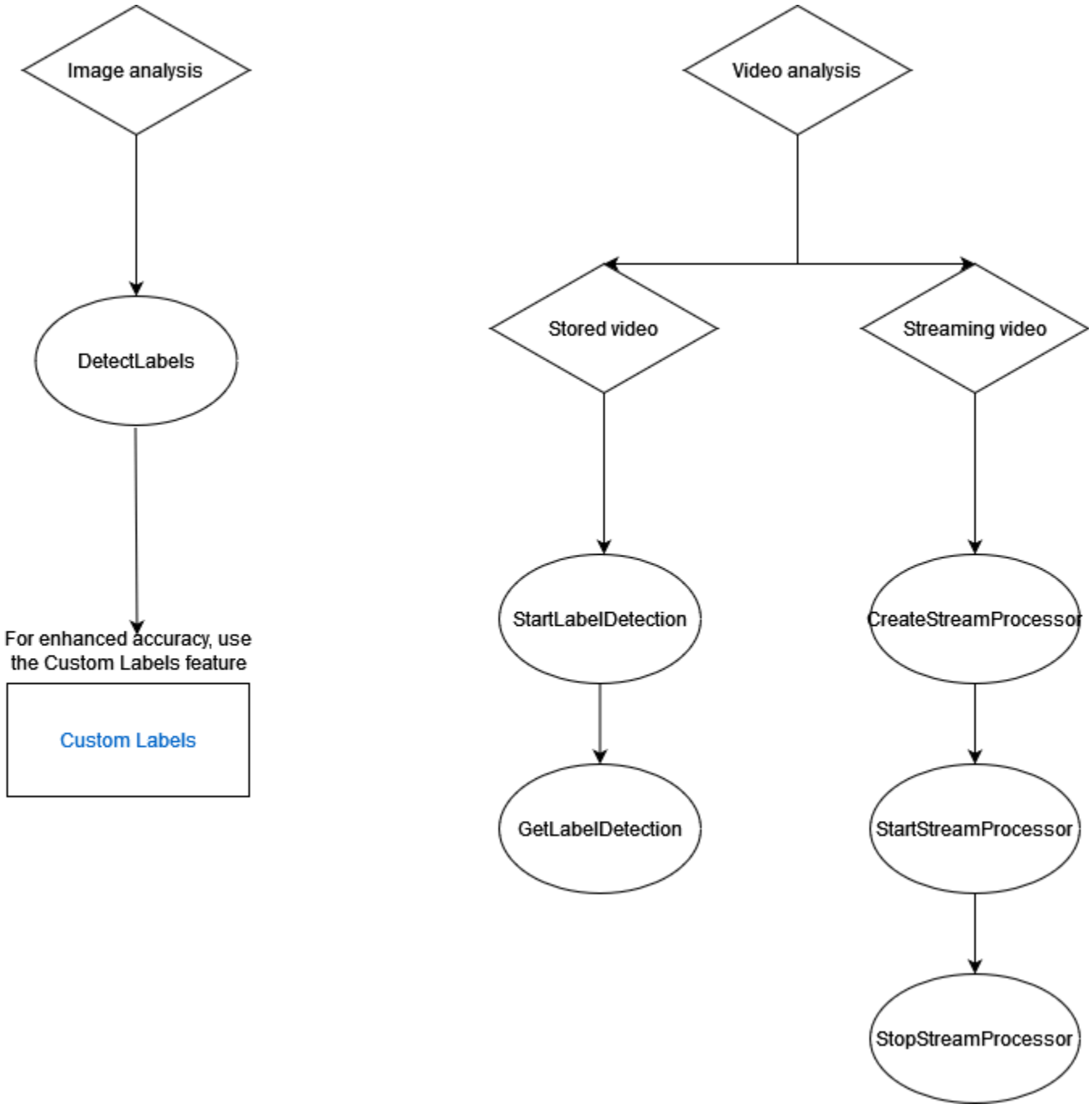
Rekognition 标签检测操作支持的标签

- 要下载 Amazon Rekognition 支持的最新标签和对象边界框列表，请单击此处。
- 要下载之前的标签和对象边界框列表，请单击此处。

> **Note**
>
> Amazon Rekognition 根据特定图像中一个人的外表进行性别二进制预测（男性、女性、女孩等）。这种预测不是为了对一个人的性别认同进行分类而设计的，您不应该使用 Amazon Rekognition 来做出这样的决定。例如，戴着长发假发和耳环扮演角色的男演员可能被预测为女性。
> 使用 Amazon Rekognition 进行性别二进制预测最适合需要在不识别特定用户的情况下分析汇总性别分布统计数据的使用案例。例如，社交媒体平台用户中女性与男性的比例。
> 我们不建议使用性别二进制预测来做出会影响个人权利、隐私或服务访问权限的决策。

Amazon Rekognition 返回英文标签。您可以使用 Amazon Translate 将英文标签翻译成其他语言。

下图显示了调用操作的顺序，具体取决于您使用 Amazon Rekognition Image 或 Amazon Rekognition Video 操作的目标：



为响应对象添加标签

边界框

Amazon Rekognition Image 和 Amazon Rekognition Video 可以为常见对象标签（例如车辆、家具、服装或宠物）返回边界框。对于不常见的对象标签，将不会返回边界框信息。您可以使用边界框找到对象在图像中的确切位置，对检测到的对象实例计数或者使用边界框尺寸衡量对象的大小。

例如，在下图中，Amazon Rekognition Image 能够检测是否存在人员、滑板、停放的车辆和其他信息。Amazon Rekognition Image 还会返回检测到的人员以及其他检测到的对象（例如汽车和车轮）的边界框。

置信度得分

Amazon Rekognition Video 和 Amazon Rekognition Image 提供了百分比分数，用于表示 Amazon Rekognition 对每个检测到的标签的准确性中具有的置信度。

父级

Amazon Rekognition Image 和 Amazon Rekognition Video 使用原级标签的分层分类对标签进行分类。例如，一个正在穿过马路的人可能会检测为行人。行人 的父标签为人。在响应中会返回这两个标签。所有原级标签将返回，给定的标签包含其父级和其他原级标签的列表。例如，祖父和曾祖父标签（如果存在）。您可以使用父标签生成一组相关标签，以允许在一个或多个图像中查询类似的标签。例如，查询所有交通工具 可能会返回一张图像中的汽车，以及另一张图像中的摩托车。

类别

Amazon Rekognition Image 和 Amazon Rekognition Video 返回标签类别的相关信息。标签是根据常见功能和上下文将各个标签组合在一起的类别的一部分，例如“车辆和汽车”和“食品和饮料”。标签类别可以是父类别的子类别。

别名

除返回标签外，Amazon Rekognition Image 和 Amazon Rekognition Video 还会返回与标签相关的所有别名。别名是具有相同含义的标签，或在视觉上可与返回的主标签互换的标签。例如，“Cell Phone”是“Mobile Phone”的别名。

在之前的版本中，Amazon Rekognition Image 在包含“Cell Phone”的主标签名称列表中返回了诸如“Mobile Phone”之类的别名。Amazon Rekognition Image 现在在名为“别名”的字段中返回“Cell Phone”，在主要标签名称列表中返回“Mobile Phone”。如果您的应用程序依赖于先前版本的 Rekognition 返回的结构，则可能需要将图像或视频标签检测操作返回的当前响应转换为之前的响应结构，其中所有标签和别名都作为主标签返回。

如果您需要将来自 DetectLabels API 的当前响应（用于图像中的标签检测）转换为之前的响应结构，请参阅中的代码示例转变 DetectLabels应对措施。

如果您需要将来自 GetLabelDetection API 的当前响应（用于存储视频中的标签检测）转换为之前的响应结构，请参阅中的代码示例转变回 GetLabelDetection 应。

图像属性

Amazon Rekognition Image 返回有关整张图片的图像质量（锐度、亮度和对比度）的信息。还会返回图像前景和背景的锐度和亮度。图像属性还可用于检测整个图像、前景、背景和带有边界框的对象的主色。

以下是正在 ImageProperties 处理的图像的 DetectLabels 操作响应中包含的数据示例：

Image Properties	Dominant Colors Examples and Pixel Percentage		Image Quality
Entire Image		Hex code #808080, RGB (128, 128, 128), 15.72	Brightness: 76.08 Sharpness: 89.72 Contrast: 88.42
		Hex code #000000, RGB (0, 0, 0), 15.10	
		Hex code #696969, RGB (105, 105, 105), 14.02	
		Hex code #8fbc8f, RGB (143, 188, 143), 12.70	
		Hex code #5f9ea0, RGB (95, 158, 160), 11.92	
Foreground		Hex code #8fbc8f, RGB (143, 188, 143), 30.18	Brightness: 79.48 Sharpness: 93.47
		Hex code #5f9ea0, RGB (95, 158, 160), 24.29	
		Hex code #000000, RGB (0, 0, 0), 12.02	
		Hex code #2f4f4f, RGB (47, 79, 79), 9.20	
		Hex code #696969, RGB (105, 105, 105), 8.95	
Background		Hex code #808080, RGB (128, 128, 128), 21.16	Brightness: 74.42 Sharpness: 87.84
		Hex code #2f4f4f, RGB (47, 79, 79), 14.61	
		Hex code #000000, RGB (0, 0, 0), 14.23	
		Hex code #696969, RGB (105, 105, 105), 13.19	
		Hex code #ffebcd, RGB (255, 235, 205), 12.80	
Car (example of objects with bounding boxes)		Hex code #5f9ea0, RGB (95, 158, 160), 29.18	Not applicable
		Hex code #8fbc8f, RGB (143, 188, 143), 14.39	
		Hex code #000000, RGB (0, 0, 0), 11.76	
		Hex code #808080, RGB (128, 128, 128), 11.38	
		Hex code #2f4f4f, RGB (47, 79, 79), 9.44	

图片属性不适用于 Amazon Rekognition Video。

模型版本

Amazon Rekognition Image 和 Amazon Rekognition Video 返回用于检测图像或存储视频中的标签的标签检测模型版本。

纳入和排除筛选器

您可以筛选 Amazon Rekognition Image 和 Amazon Rekognition Video 标签检测操作返回的结果。通过提供标签和类别的筛选标准来筛选结果。标签筛选器可以执行纳入或排除操作。

有关使用 `DetectLabels` 筛选获得的结果的更多信息，请参阅检测图像中的标签。

有关使用 `GetLabelDetection` 筛选获得的结果的更多信息，请参阅检测视频中的标签。

对结果进行排序和汇总

从某些 Amazon Rekognition Video 操作中获得的结果可以根据时间戳和视频片段进行排序和汇总。分别使用 GetLabelDetection 或 GetContentModeration 检索标签检测或内容审核作业的结果时，您可以使用 SortBy 和 AggregateBy 参数来指定希望如何返回结果。您可以SortBy与TIMESTAMP或NAME（标签名称）一起使用，也可以将TIMESTAMPS或SEGMENTS与AggregateBy参数一起使用。

检测图像中的标签

您可以使用该DetectLabels操作来检测图像中的标签（对象和概念），并检索有关图像属性的信息。图像属性包括诸如前景和背景的颜色以及图像的锐度、亮度和对比度之类的属性。您可以只检索图像中的标签，也可以仅检索图像的属性，或者两者兼而有之。有关示例，请参阅分析存储在 Amazon S3 存储桶中的图像。

以下示例使用各种 AWS SDKs 和 to AWS CLI call DetectLabels。有关 DetectLabels 操作响应的信息，请参阅DetectLabels 响应。

检测图像中的标签

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将其中包含一个或多个对象（如树木、房屋和船）的图像上传到您的 S3 存储桶。图像的格式必须为 .jpg 或 .png。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 DetectLabels 操作。

   Java

   此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
package com.amazonaws.samples;
import java.util.List;

import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.Instance;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.Parent;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;

public class DetectLabels {

    public static void main(String[] args) throws Exception {

        String photo = "photo";
        String bucket = "bucket";

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image().withS3Object(new
 S3Object().withName(photo).withBucket(bucket)))
                .withMaxLabels(10).withMinConfidence(75F);

        try {
            DetectLabelsResult result = rekognitionClient.detectLabels(request);
            List<Label> labels = result.getLabels();

            System.out.println("Detected labels for " + photo + "\n");
            for (Label label : labels) {
                System.out.println("Label: " + label.getName());
                System.out.println("Confidence: " +
 label.getConfidence().toString() + "\n");

                List<Instance> instances = label.getInstances();
                System.out.println("Instances of " + label.getName());
                if (instances.isEmpty()) {
                    System.out.println("  " + "None");
```

```
                } else {
                    for (Instance instance : instances) {
                        System.out.println("  Confidence: " +
 instance.getConfidence().toString());
                        System.out.println("  Bounding box: " +
 instance.getBoundingBox().toString());
                    }
                }
                System.out.println("Parent labels for " + label.getName() +
 ":");
                List<Parent> parents = label.getParents();
                if (parents.isEmpty()) {
                    System.out.println("  None");
                } else {
                    for (Parent parent : parents) {
                        System.out.println("  " + parent.getName());
                    }
                }
                System.out.println("--------------------");
                System.out.println();

            }
        } catch (AmazonRekognitionException e) {
            e.printStackTrace();
        }
    }
}
```

AWS CLI

此示例显示 detect-labels CLI 操作的 JSON 输出。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将profile-name的值替换为您的开发人员资料的名称。

```
aws rekognition detect-labels --image '{ "S3Object": { "Bucket": "bucket-name",
 "Name": "file-name" } }' \
--features GENERAL_LABELS IMAGE_PROPERTIES \
--settings '{"ImageProperties": {"MaxDominantColors":1}, {"GeneralLabels":
{"LabelInclusionFilters":["Cat"]}}}' \
--profile profile-name \
--region us-east-1
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition detect-labels --image "{\"S3Object\":{\"Bucket\":\"bucket-name
\",\"Name\":\"file-name\"}}" --features GENERAL_LABELS IMAGE_PROPERTIES \
--settings "{\"GeneralLabels\":{\"LabelInclusionFilters\":[\"Car\"]}}" --profile
 profile-name --region us-east-1
```

Python

此示例显示在输入图像中检测到的标签。在函数 `main` 中，将`bucket`和`photo`的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_labels(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
    MaxLabels=10,
    # Uncomment to use image properties and filtration settings
    #Features=["GENERAL_LABELS", "IMAGE_PROPERTIES"],
    #Settings={"GeneralLabels": {"LabelInclusionFilters":["Cat"]},
    # "ImageProperties": {"MaxDominantColors":10}}
    )

    print('Detected labels for ' + photo)
    print()
    for label in response['Labels']:
        print("Label: " + label['Name'])
        print("Confidence: " + str(label['Confidence']))
```

```
        print("Instances:")

        for instance in label['Instances']:
            print(" Bounding box")
            print(" Top: " + str(instance['BoundingBox']['Top']))
            print(" Left: " + str(instance['BoundingBox']['Left']))
            print(" Width: " + str(instance['BoundingBox']['Width']))
            print(" Height: " + str(instance['BoundingBox']['Height']))
            print(" Confidence: " + str(instance['Confidence']))
            print()

        print("Parents:")
        for parent in label['Parents']:
           print(" " + parent['Name'])

        print("Aliases:")
        for alias in label['Aliases']:
            print(" " + alias['Name'])

            print("Categories:")
        for category in label['Categories']:
            print(" " + category['Name'])
            print("----------")
            print()

     if "ImageProperties" in str(response):
        print("Background:")
        print(response["ImageProperties"]["Background"])
        print()
        print("Foreground:")
        print(response["ImageProperties"]["Foreground"])
        print()
        print("Quality:")
        print(response["ImageProperties"]["Quality"])
        print()

     return len(response['Labels'])

def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    label_count = detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))
```

```
if __name__ == "__main__":
    main()
```

.NET

此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F
        };

        try
        {
```

```
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Ruby

此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
  # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket = 'bucket' # the bucket name without s3://
  photo  = 'photo' # the name of file
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    image: {
      s3_object: {
        bucket: bucket,
        name: photo
      },
    },
    max_labels: 10
  }
 response = client.detect_labels attrs
 puts "Detected labels for: #{photo}"
 response.labels.each do |label|
   puts "Label:      #{label.name}"
```

```
    puts "Confidence: #{label.confidence}"
    puts "Instances:"
    label['instances'].each do |instance|
      box = instance['bounding_box']
      puts "  Bounding box:"
      puts "    Top:        #{box.top}"
      puts "    Left:       #{box.left}"
      puts "    Width:      #{box.width}"
      puts "    Height:     #{box.height}"
      puts "  Confidence: #{instance.confidence}"
    end
    puts "Parents:"
    label.parents.each do |parent|
      puts "  #{parent.name}"
    end
    puts "------------"
    puts ""
  end
```

Node.js

此示例显示在输入图像中检测到的标签的列表。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

如果您使用的是 TypeScript 定义，则可能需要使用import AWS from 'aws-sdk'而不是const AWS = require('aws-sdk')，以便使用 Node.js 运行该程序。您可以查阅适用于 JavaScript 的AWS SDK，了解更多详情。根据您的配置设置方式，您可能还需要使用AWS.config.update({region:*region*});来指定您的区域。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucketname without s3://
const photo  = 'image-name' // the name of file

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

const client = new AWS.Rekognition();
```

```
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
  MaxLabels: 10
}
client.detectLabels(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // if an error occurred
  } else {
    console.log(`Detected labels for: ${photo}`)
    response.Labels.forEach(label => {
      console.log(`Label:      ${label.Name}`)
      console.log(`Confidence: ${label.Confidence}`)
      console.log("Instances:")
      label.Instances.forEach(instance => {
        let box = instance.BoundingBox
        console.log("  Bounding box:")
        console.log(`    Top:        ${box.Top}`)
        console.log(`    Left:       ${box.Left}`)
        console.log(`    Width:      ${box.Width}`)
        console.log(`    Height:     ${box.Height}`)
        console.log(`  Confidence: ${instance.Confidence}`)
      })
      console.log("Parents:")
      label.Parents.forEach(parent => {
        console.log(`  ${parent.Name}`)
      })
      console.log("------------")
      console.log("")
    }) // for response.labels
  } // if
});
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectLabels {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,ImageBucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
```

```
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

        try {
            S3Object s3Object = S3Object.builder()
                .bucket(bucket)
                .name(image)
                .build() ;

            Image myImage = Image.builder()
                .s3Object(s3Object)
                .build();

            DetectLabelsRequest detectLabelsRequest =
DetectLabelsRequest.builder()
                .image(myImage)
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
label.confidence().toString());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
// snippet-end:[rekognition.java2.detect_labels.main]
```

```
}
```

DetectLabels 操作请求

对 DetectLabel 的输入是一个图像。在此示例 JSON 输入中，源图像从 Amazon S3 存储桶加载。MaxLabels 是要在响应中返回的标签的最大数量。MinConfidence 是 Amazon Rekognition Image 对检测到的标签要在响应中返回而对其准确度所具有的最小置信度。

“功能”允许您指定要返回的图像中的一个或多个特征，允许您选择GENERAL_LABELS和IMAGE_PROPERTIES。包含GENERAL_LABELS将返回在输入图像中检测到的标签，而包含IMAGE_PROPERTIES则允许您查看图像的颜色和质量。

“设置”允许您筛选GENERAL_LABELS和IMAGE_PROPERTIES功能的返回项目。对于标签，您可以使用纳入和排除筛选器。您也可以按特定标签、单个标签或标签类别进行筛选：

- LabelInclusionFilters -允许您指定要在响应中包含哪些标签。
- LabelExclusionFilters -允许您指定要从响应中排除哪些标签。
- LabelCategoryInclusionFilters -允许您指定要在响应中包含哪些标签类别。
- LabelCategoryExclusionFilters -允许您指定要从响应中排除哪些标签类别。

您还可以根据需要组合纳入和排除筛选器，排除某些标签或类别，而纳入其他标签或类别。

IMAGE_PROPERTIES指图像的主色和质量属性，例如锐度、亮度和对比度。检测IMAGE_PROPERTIES时，您可以使用 MaxDominantColors 参数指定要返回的最大主色数（默认值为 10）。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75,
    "Features": [ "GENERAL_LABELS", "IMAGE_PROPERTIES" ],
    "Settings": {
```

```
        "GeneralLabels": {
            "LabelInclusionFilters": [<Label(s)>],
            "LabelExclusionFilters": [<Label(s)>],
            "LabelCategoryInclusionFilters": [<Category Name(s)>],
            "LabelCategoryExclusionFilters": [<Category Name(s)>]
        },
        "ImageProperties": {
            "MaxDominantColors":10
        }
    }
}
```

DetectLabels 响应

来自 DetectLabels 的响应是在图像中检测到的一组标签和检测标签时所依据的置信度级别。

以下是来自 DetectLabels 的示例响应。下面的示例响应包含为 GENERAL_LABELS 返回的各种属性，包括：

- 名称 – 检测到的标签的名称。在此示例中，操作检测到一个标有“Mobile Phone”标签的对象。
- 置信度 – 每个标签均有一个关联的置信度级别。在此示例中，标签的置信度为 99.36%。
- 父级 – 检测到的标签的原级标签。在此示例中，“Mobile Phone”标签有一个名为“Phone”的父标签。
- 别名 – 有关标签可能的别名的信息。在此示例中，“Mobile Phone”标签的别名可能是“Cell Phone”。
- 类别 – 检测到的标签所属的标签类别。在此示例中，它是“技术和计算”。

常见对象标签的响应包括边界框信息，针对输入图像上标签的位置。例如，“人”标签有包含两个边界框的实例数组。这两个边界框是在图像中检测到的两个人的位置。

响应还包括与 IMAGE_PROPERTIES 相关的属性。IMAGE_PROPERTIES 功能提供的属性是：

- 质量 – 有关输入图像锐度、亮度和对比度的信息，得分介于 0 到 100 之间。报告整张图像以及图像背景和前景的质量（如果有）。但是，仅报告整张图像的对比度，而背景和前景也会报告锐度和亮度。
- 主色 – 图像中占主导地位的颜色数组。每种主色均使用简化的颜色名称、CSS 调色板、RGB 值和十六进制代码进行描述。
- 前景 – 有关输入图像前景的主色彩、锐度和亮度的信息。
- 背景 – 有关输入图像背景的主色彩、锐度和亮度的信息。

当 GENERAL_LABELS 和 IMAGE_PROPERTIES 一起用作输入参数时，Amazon Rekognition Image 还将返回带有边界框的对象的主色。

字段 LabelModelVersion 包含由 DetectLabels 使用的检测模型的版本号。


```
{

   "Labels": [
        {
            "Name": "Mobile Phone",
            "Parents": [
              {
                "Name": "Phone"
              }
            ],
            "Aliases": [
              {
                "Name": "Cell Phone"
              }
            ],
            "Categories": [
              {
                "Name": "Technology and Computing"
              }
            ],
            "Confidence": 99.9364013671875,
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.26779675483703613,
                        "Height": 0.8562285900115967,
                        "Left": 0.3604024350643158,
                        "Top": 0.09245597571134567,
                    }
                    "Confidence": 99.9364013671875,
                    "DominantColors": [
                    {
                "Red": 120,
                "Green": 137,
                "Blue": 132,
                "HexCode": "3A7432",
                "SimplifiedColor": "red",
                "CssColor": "fuscia",
                "PixelPercentage": 40.10
```

```
                    }
                        ],
                }
            ]
        }
    ],
    "ImageProperties": {
        "Quality": {
            "Brightness": 40,
            "Sharpness": 40,
            "Contrast": 24,
        },
        "DominantColors": [
            {
                "Red": 120,
                "Green": 137,
                "Blue": 132,
                "HexCode": "3A7432",
                "SimplifiedColor": "red",
                "CssColor": "fuscia",
                "PixelPercentage": 40.10
            }
        ],
        "Foreground": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "red",
                    "PixelPercentage": 30.70
                }
            ],
        }
        "Background": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
```

```
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "Red",
                    "PixelPercentage": 10.20
                }
            ],
        },
    },
    "LabelModelVersion": "3.0"
}
```

转变 DetectLabels应对措施

使用 DetectLabels API 时，您可能需要响应结构来模仿旧的 API 响应结构，其中主标签和别名都包含在同一个列表中。

以下是来自的当前 API 响应的示例 DetectLabels：

```
"Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 99.99717712402344,
            "Instances": [],
            "Parents": [
                {
                "Name": "Phone"
                }
             ],
            "Aliases": [
                {
                "Name": "Cell Phone"
                }
             ]
        }
 ]
```

以下示例显示了 DetectLabelsAPI 之前的响应：

```
"Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 99.99717712402344,
            "Instances": [],
            "Parents": [
                {
                "Name": "Phone"
                }
             ]
         },
         {
            "Name": "Cell Phone",
            "Confidence": 99.99717712402344,
            "Instances": [],
            "Parents": [
                {
                "Name": "Phone"
                }
             ]
         },
]
```

如果需要，您可以转换当前响应以遵循旧响应的格式。您可以使用以下示例代码将最新的 API 响应转换为之前的 API 响应结构：

Python

以下代码示例演示了如何转换来自 DetectLabels API 的当前响应。在下面的代码示例中，您可以将的*EXAMPLE_INFERENCE_OUTPUT*值替换为您已运行的 DetectLabels 操作的输出。

```
from copy import deepcopy

LABEL_KEY = "Labels"
ALIASES_KEY = "Aliases"
INSTANCE_KEY = "Instances"
NAME_KEY = "Name"

#Latest API response sample
EXAMPLE_INFERENCE_OUTPUT = {
    "Labels": [
```

```
        {
            "Name": "Mobile Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Aliases": [
                {
                    "Name": "Cell Phone"
                }
            ],
            "Instances":[
                {
                    "BoundingBox":{
                        "Height":0.1549897,
                        "Width":0.07747964,
                        "Top":0.50858885,
                        "Left":0.00018205095
                    },
                    "Confidence":98.401276
                }
            ]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}

def expand_aliases(inferenceOutputsWithAliases):

    if LABEL_KEY in inferenceOutputsWithAliases:
        expandInferenceOutputs = []
        for primaryLabelDict in inferenceOutputsWithAliases[LABEL_KEY]:
            if ALIASES_KEY in primaryLabelDict:
                for alias in primaryLabelDict[ALIASES_KEY]:
                    aliasLabelDict = deepcopy(primaryLabelDict)
                    aliasLabelDict[NAME_KEY] = alias[NAME_KEY]
```

```
                    del aliasLabelDict[ALIASES_KEY]
                    if INSTANCE_KEY in aliasLabelDict:
                        del aliasLabelDict[INSTANCE_KEY]
                    expandInferenceOutputs.append(aliasLabelDict)

        inferenceOutputsWithAliases[LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases


if __name__ == "__main__":

    outputWithExpandAliases = expand_aliases(EXAMPLE_INFERENCE_OUTPUT)
    print(outputWithExpandAliases)
```

下面是转换后的响应示例：

```
#Output example after the transformation
{
    "Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Aliases": [
                {
                    "Name": "Cell Phone"
                }
            ],
            "Instances":[
                {
                    "BoundingBox":{
                        "Height":0.1549897,
                        "Width":0.07747964,
                        "Top":0.50858885,
                        "Left":0.00018205095
                    },
                    "Confidence":98.401276
```

```
                }
            ]
        },
        {
            "Name": "Cell Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Instances":[]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}
```

检测视频中的标签

Amazon Rekognition Video 可以检测视频中的标签（对象和概念）以及检测到标签的时间。有关开发工具包代码示例，请参阅使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。有关 AWS CLI 示例，请参阅使用分析视频 AWS Command Line Interface。

Amazon Rekognition Video 标签检测是一项异步操作。要开始检测视频中的标签，请致电StartLabelDetection。

Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service 主题。如果视频分析成功，请调用 GetLabelDetection 来获取检测到的标签。有关调用视频分析 API 操作的信息，请参阅调用 Amazon Rekognition Video 操作。

StartLabelDetection请求

以下示例是 StartLabelDetection 操作的请求。您为 StartLabelDetection 操作提供了存储在 Amazon S3 存储桶中的视频。在示例请求 JSON 中，指定了 Amazon S3 存储桶和视频名称，以及 MinConfidence、Features、Settings 和 NotificationChannel。

MinConfidence 是 Amazon Rekognition Video 对检测到的标签或实例边界框（如果检测到）要在响应中返回而对其准确度所具有的最小置信度。

使用 Features，您可以指定要将 GENERAL_LABELS 作为响应的一部分返回。

使用 Settings，您可以筛选 GENERAL_LABELS 的返回项目。对于标签，您可以使用纳入和排除筛选器。您也可以按特定标签、单个标签或标签类别进行筛选：

- LabelInclusionFilters – 用于指定要在响应中纳入哪些标签
- LabelExclusionFilters – 用于指定要从响应中排除哪些标签。
- LabelCategoryInclusionFilters – 用于指定要在响应中纳入哪些标签类别。
- LabelCategoryExclusionFilters - 用于指定要从响应中排除哪些标签类别。

您还可以根据需要组合纳入和排除筛选器，排除某些标签或类别，而纳入其他标签或类别。

NotificationChannel 是您希望 Amazon Rekognition Video 向其发布标签检测操作完成状态的 Amazon SNS 主题的 ARN。如果您使用的是 AmazonRekognitionServiceRole 权限策略，那么 Amazon SNS 主题的主题名称必须以 Rekognition 开头。

以下是 JSON 格式的示例 StartLabelDetection 请求，包括筛选器：

```
{
    "ClientRequestToken": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "JobTag": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
         }
     },
     "Features": ["GENERAL_LABELS"],
     "MinConfidence": 75,
     "Settings": {
         "GeneralLabels": {
             "LabelInclusionFilters": ["Cat", "Dog"],
```

```
            "LabelExclusionFilters": ["Tiger"],
            "LabelCategoryInclusionFilters": ["Animals and Pets"],
            "LabelCategoryExclusionFilters": ["Popular Landmark"]
        }
    },
    "NotificationChannel": {
        "RoleArn": "arn:aws:iam::012345678910:role/SNSAccessRole",
        "SNSTopicArn": "arn:aws:sns:us-east-1:012345678910:notification-topic",
    }
}
```

GetLabelDetection 操作响应

GetLabelDetection 将返回一个数组 (Labels)，其中包含有关在视频中检测到的标签的信息。数组可以按时间排序，也可以按指定 SortBy 参数时检测到的标签进行排序。也可以使用 AggregateBy 参数选择如何汇总响应项。

以下示例是 GetLabelDetection 的 JSON 响应。在响应中，请注意以下内容：

- 排序顺序 – 返回的标签数组按时间进行排序。要按标签进行排序，请为 GetLabelDetection 在 SortBy 输入参数中指定 NAME。如果此标签在视频中多次出现，则会有 (LabelDetection) 元素的多个实例。默认排序顺序为 TIMESTAMP，而辅助排序顺序为 NAME。
- 标签信息 – LabelDetection 数组元素包含一个（ 标签 ）对象，该对象包含标签名称和 Amazon Rekognition 在检测到的标签的准确性中具有的置信度。Label 对象还包括标签的分层分类和常见标签的边界框信息。Timestamp 是从视频开始到检测到标签的时间，以毫秒为单位。

  还会返回与标签关联的任何类别或别名的相关信息。对于按视频SEGMENTS汇总的结果，将返回 StartTimestampMillis、EndTimestampMillis 和 DurationMillis 结构，它们分别定义了片段的开始时间、结束时间和持续时间。
- 汇总 – 指定返回结果时的汇总方式。默认为按 TIMESTAMPS 汇总。您也可以选择按 SEGMENTS 汇总，即在某个时间段内汇总结果。如果按 SEGMENTS 汇总，则不会返回有关检测到的带有边界框的实例的信息。只返回在分段期间检测到的标签。
- 分页信息 - 此示例显示一页标签检测信息。您可以为 GetLabelDetection 在 MaxResults 输入参数中指定要返回的 LabelDetection 对象的数量。如果存在的结果的数量超过了 MaxResults，则 GetLabelDetection 会返回一个令牌 (NextToken)，用于获取下一页的结果。有关更多信息，请参阅 获取 Amazon Rekognition Video 分析结果。
- 视频信息 – 此响应包含有关由 GetLabelDetection 返回的每页信息中的视频格式（ VideoMetadata ）的信息。

以下是 JSON 格式的示例 GetLabelDetection 响应，其中包含由 TIMESTAMPS 进行聚合：

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "Timestamp": 1000,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875, // Classification confidence
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567
                        },
                        "Confidence": 99.9364013671875 // Detection confidence
                    }
                ]
            }
        },
        {
            "Timestamp": 1000,
            "Label": {
                "Name": "Cup",
                "Categories": [
```

```
                {
                    "Name": "Kitchen and Dining"
                }
            ],
            "Aliases": [
                {
                    "Name": "Mug"
                }
            ],
            "Parents": [],
            "Confidence": 99.9364013671875, // Classification confidence
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.26779675483703613,
                        "Height": 0.8562285900115967,
                        "Left": 0.3604024350643158,
                        "Top": 0.09245597571134567
                    },
                    "Confidence": 99.9364013671875 // Detection confidence
                }
            ]
        }
    },
    {
        "Timestamp": 2000,
        "Label": {
            "Name": "Kangaroo",
            "Categories": [
                {
                    "Name": "Animals and Pets"
                }
            ],
            "Aliases": [
                {
                    "Name": "Wallaby"
                }
            ],
            "Parents": [
                {
                    "Name": "Mammal"
                }
            ],
            "Confidence": 99.9364013671875,
```

```
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567,
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
            }
        },
        {
            "Timestamp": 4000,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
```

```
            }
        }
    ],
    "VideoMetadata": {
        "ColorRange": "FULL",
        "DurationMillis": 5000,
        "Format": "MP4",
        "FrameWidth": 1280,
        "FrameHeight": 720,
        "FrameRate": 24
    }
}
```

以下是 JSON 格式的示例 GetLabelDetection 响应，其中包含按分段进行聚合：

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "StartTimestampMillis": 225,
            "EndTimestampMillis": 3578,
            "DurationMillis": 3353,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875 // Maximum confidence score for Segment
 mode
            }
```

```
        },
        {
            "StartTimestampMillis": 7578,
            "EndTimestampMillis": 12371,
            "DurationMillis": 4793,
            "Label": {
                "Name": "Kangaroo",
                "Categories": [
                  {
                    "Name": "Animals and Pets"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Wallaby"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Mammal"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        },
        {
            "StartTimestampMillis": 22225,
            "EndTimestampMillis": 22578,
            "DurationMillis": 2353,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
```

```
                }
            ],
            "Confidence": 99.9364013671875
        }
    }
],
"VideoMetadata": {
    "ColorRange": "FULL",
    "DurationMillis": 5000,
    "Format": "MP4",
    "FrameWidth": 1280,
    "FrameHeight": 720,
    "FrameRate": 24
}
}
```

转变回 GetLabelDetection 应

使用 GetLabelDetection API 操作检索结果时，您可能需要响应结构来模仿旧的 API 响应结构，其中主标签和别名都包含在同一个列表中。

上一节中的 JSON 响应示例，显示了来自的 API 响应的当前形式 GetLabelDetection。

以下示例显示了 GetLabelDetection API 之前的响应：

```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Leaf"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
```

```
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.11109819263219833,
                            "Top": 0.08098889887332916,
                            "Left": 0.8881205320358276,
                            "Height": 0.9073750972747803
                        },
                        "Confidence": 99.5831298828125
                    },
                    {
                        "BoundingBox": {
                            "Width": 0.1268676072359085,
                            "Top": 0.14018426835536957,
                            "Left": 0.0003282368124928324,
                            "Height": 0.7993982434272766
                        },
                        "Confidence": 99.46029663085938
                    }
                ],
                "Confidence": 99.63411102294922,
                "Parents": [],
                "Name": "Person"
            }
        },
        .
        .
        .

        {
            "Timestamp": 166,
            "Label": {
                "Instances": [],
                "Confidence": 73.6471176147461,
                "Parents": [
                    {
                        "Name": "Clothing"
                    }
                ],
```

```
                "Name": "Sleeve"
            }
        }

    ],
    "LabelModelVersion": "2.0",
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Format": "QuickTime / MOV",
        "FrameRate": 23.976024627685547,
        "Codec": "h264",
        "DurationMillis": 5005,
        "FrameHeight": 674,
        "FrameWidth": 1280
    }
}
```

如果需要，您可以转换当前响应以遵循旧响应的格式。您可以使用以下示例代码将最新的 API 响应转换为之前的 API 响应结构：

```
from copy import deepcopy

VIDEO_LABEL_KEY = "Labels"
LABEL_KEY = "Label"
ALIASES_KEY = "Aliases"
INSTANCE_KEY = "Instances"
NAME_KEY = "Name"

#Latest API response sample for AggregatedBy SEGMENTS
EXAMPLE_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
```

```
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],

            },
            "StartTimestampMillis": 6400,
            "EndTimestampMillis": 8200,
            "DurationMillis": 1800
        },
    ]
}

#Output example after the transformation for AggregatedBy SEGMENTS
EXPECTED_EXPANDED_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
```

```
                {
                    "Name": "Human"
                },
            ],
            "Categories": [
                {
                    "Name": "Person Description"
                }
            ],
        },
        "StartTimestampMillis": 0,
        "EndTimestampMillis": 500666,
        "DurationMillis": 500666
    },
    {
        "Timestamp": 6400,
        "Label": {
            "Name": "Leaf",
            "Confidence": 89.77790069580078,
            "Parents": [
                {
                    "Name": "Plant"
                }
            ],
            "Aliases": [],
            "Categories": [
                {
                    "Name": "Plants and Flowers"
                }
            ],

        },
        "StartTimestampMillis": 6400,
        "EndTimestampMillis": 8200,
        "DurationMillis": 1800
    },
    {
        "Timestamp": 0,
        "Label":{
            "Name": "Human",
            "Confidence": 97.530106,
            "Parents": [],
            "Categories": [
                {
```

```
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
    ]
}

#Latest API response sample for AggregatedBy TIMESTAMPS
EXAMPLE_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
```

```
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
    ]
}

#Output example after the transformation for AggregatedBy TIMESTAMPS
EXPECTED_EXPANDED_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
```

```
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Human",
                "Confidence": 97.530106,
                "Parents": [],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
    ]
}
```

```
def expand_aliases(inferenceOutputsWithAliases):

    if VIDEO_LABEL_KEY in inferenceOutputsWithAliases:
        expandInferenceOutputs = []
        for segmentLabelDict in inferenceOutputsWithAliases[VIDEO_LABEL_KEY]:
            primaryLabelDict = segmentLabelDict[LABEL_KEY]
            if ALIASES_KEY in primaryLabelDict:
                for alias in primaryLabelDict[ALIASES_KEY]:
                    aliasLabelDict = deepcopy(segmentLabelDict)
                    aliasLabelDict[LABEL_KEY][NAME_KEY] = alias[NAME_KEY]
                    del aliasLabelDict[LABEL_KEY][ALIASES_KEY]
                    if INSTANCE_KEY in aliasLabelDict[LABEL_KEY]:
                        del aliasLabelDict[LABEL_KEY][INSTANCE_KEY]
                    expandInferenceOutputs.append(aliasLabelDict)

        inferenceOutputsWithAliases[VIDEO_LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases


if __name__ == "__main__":

    segmentOutputWithExpandAliases = expand_aliases(EXAMPLE_SEGMENT_OUTPUT)
    assert segmentOutputWithExpandAliases == EXPECTED_EXPANDED_SEGMENT_OUTPUT

    timestampOutputWithExpandAliases = expand_aliases(EXAMPLE_TIMESTAMP_OUTPUT)
    assert timestampOutputWithExpandAliases == EXPECTED_EXPANDED_TIMESTAMP_OUTPUT
```

检测流视频事件中的标签

您可以使用 Amazon Rekognition Video 来检测流视频中的标签。为此，您需要创建一个流处理器 (CreateStreamProcessor) 来启动和管理对流视频的分析。

Amazon Rekognition Video 使用 Amazon Kinesis Video Streams 来接收和处理视频流。创建流处理器时，可选择希望流处理器检测的内容。您可以选择人员、包裹和宠物，也可以选择人员和包裹。分析结果将输出到 Amazon S3 存储桶和 Amazon SNS 通知中。请注意，Amazon Rekognition Video 会检测到视频中有人在场，但不会检测到该人是否是特定的个人。从流视频的集合中搜索人脸，请参阅the section called “在流视频中搜索集合中的人脸”。

要将 Amazon Rekognition Video 与流视频配合使用，您的应用程序需要满足以下条件：

- 用于向 Amazon Rekognition Video 发送流视频的 Kinesis 视频流。有关更多信息，请参阅 Amazon Kinesis Video Streams 开发人员指南。
- 一个 Amazon Rekognition Video 流处理器，用于管理对流视频的分析。有关更多信息，请参阅 Amazon Rekognition Video 流处理器操作概述。
- Amazon S3 存储桶。Amazon Rekognition Video 将会话输出发布到 S3 存储桶。输出包括首次检测到感兴趣的人或物体的图像帧。您必须是 S3 存储桶拥有者。
- 亚马逊 Rekognition Video 向其发布智能提醒和摘要的亚马逊 SNS 主题。 end-of-session

主题

- 设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源
- 调用流视频事件的标签检测操作

设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源

以下过程描述了配置 Kinesis 视频流以及用于检测流视频中标签的其他资源的步骤。

先决条件

要运行此过程， 适用于 Java 的 AWS SDK 必须安装。有关更多信息，请参阅 Amazon Rekognition 入门。您使用的 AWS 账户需要具有对 Amazon Rekognition API 的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 Amazon Rekognition 定义的操作。

检测视频流中的标签 (AWS SDK)

1. 创建 Amazon S3 存储桶。记下要使用的存储桶名称和各种键前缀。稍后您将使用此信息。
2. 创建 Amazon SNS 主题。当在视频流中首次检测到感兴趣的物体时，可使用它接收通知。记下该主题的 Amazon 资源名称 (ARN)。有关更多信息，请参阅《Amazon SNS 开发人员指南》中的创建 Amazon SNS 主题。
3. 为端点订阅 Amazon SNS 主题。有关更多信息，请参阅《Amazon SNS 开发人员指南》中的订阅 Amazon SNS 主题。
4. 创建 Kinesis 视频流并记下流的 Amazon 资源名称 (ARN)。
5. 如果您还没有这样做，请创建一个 IAM 服务角色，让 Amazon Rekognition Video 可以访问您的 Kinesis 视频流、S3 存储桶和 Amazon SNS 主题。有关更多信息，请参阅 为标签检测流处理器提供访问权限。

然后，您可以使用所选的流处理器名称创建标签检测流处理器并启动流处理器。

> **Note**
>
> 只有在确认可以将媒体摄取到 Kinesis 视频流之后，才启动流处理器。

摄像机方向和设置

Amazon Rekognition Video 流视频事件可以支持 Kinesis Video Streams 支持的所有摄像机。为获得最佳效果，我们建议将摄像机放置在距离地面 0 到 45 度之间。摄像机必须处于标准的直立位置。例如，如果画面中有一个人，这个人的方向应该是垂直的，人的头部在画面中应该高于脚部。

为标签检测流处理器提供访问权限

您可以使用 AWS Identity and Access Management (IAM) 服务角色向亚马逊 Rekognition Video 授予 Kinesis 视频流的读取权限。为此，请使用 IAM 角色授予 Amazon Rekognition Video 访问您的 Amazon S3 存储桶和 Amazon SNS 主题的权限。

您可以创建权限策略，允许 Amazon Rekognition Video 访问现有 Amazon SNS 主题、Amazon S3 存储桶和 Kinesis 视频流。有关使用 step-by-step过程的信息 AWS CLI，请参见the section called “AWS CLI 设置标签检测 IAM 角色的命令”。

授予 Amazon Rekognition Video 访问标签检测资源的权限

1. 使用 IAM JSON 策略编辑器创建新的权限策略，然后使用以下策略。将 `kvs-stream-name`替换为 Kinesis 视频流的名称、将 `topicarn` 替换为您要使用的 Amazon SNS 主题的 Amazon 资源名称 (ARN)，以及将`bucket-name`替换为 Amazon S3 存储桶的名称。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisVideoPermissions",
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": [
```

```
                "arn:aws:kinesisvideo:::stream/kvs-stream-name/*"
            ]
        },
        {
            "Sid": "SNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:::sns-topic-name"
            ]
        },
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

2. 创建 IAM 服务角色，或者更新现有 IAM 服务角色。使用以下信息创建 IAM 服务角色：

   1. 对于服务名称，选择 Rekognition。
   2. 对于服务角色使用案例，选择 Rekognition。
   3. 附加您在步骤 1 中创建的权限策略。

3. 记下服务角色的 ARN。在执行视频分析操作之前，您需要使用它来创建流处理器。

4. （可选）如果您使用自己的 AWS KMS 密钥对发送到 S3 存储桶的数据进行加密，则必须添加带有 IAM 角色的以下语句。（这是您为密钥政策创建的 IAM 角色，它与要使用的客户托管密钥相对应。）

```
            {
                        "Sid": "Allow use of the key by label detection Role",
                        "Effect": "Allow",
```

```
                        "Principal": {
                            "AWS":
 "arn:aws:iam:::role/REPLACE_WITH_LABEL_DETECTION_ROLE_CREATED"
                        },
                        "Action": [
                            "kms:Decrypt",
                            "kms:GenerateDataKey*"
                        ],
                        "Resource": "*"
            }
```

AWS CLI 设置标签检测 IAM 角色的命令

如果您还没有，请 AWS CLI 使用您的凭据进行设置和配置。

在中输入以下命令 AWS CLI 以设置具有标签检测所需权限的 IAM 角色。

1. `export IAM_ROLE_NAME=labels-test-role`
2. `export AWS_REGION=us-east-1`
3. 创建包含以下内容的信任关系策略文件（例如 assume-role-rekognition .json）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rekognition.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. `aws iam create-role --role-name $IAM_ROLE_NAME --assume-role-policy-document file://path-to-assume-role-rekognition.json --region $AWS_REGION`

5. `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/service-role/AmazonRekognitionServiceRole" --region $AWS_REGION`
6. 如果您想要接收通知的 SNS 主题名称不以 “AmazonRekognition” 前缀开头，请添加以下政策：

   `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/AmazonSNSFullAccess" --region $AWS_REGION`
7. 如果您使用自己的 AWS KMS 密钥加密发送到 Amazon S3 存储桶的数据，请更新要使用的客户托管密钥的密钥政策。

   a. 创建一个包含以下内容的文件 kms_key_policy.json：

   ```
   {
   "Sid": "Allow use of the key by label detection Role",
   "Effect": "Allow",
   "Principal": {
   "AWS": "arn:aws:iam:::role/REPLACE_WITH_IAM_ROLE_NAME_CREATED"
   },
   "Action": [
   "kms:Encrypt",
   "kms:GenerateDataKey*"
   ],
   "Resource": "*"
   }
   ```

   b. `export KMS_KEY_ID=labels-kms-key-id`. 将 KMS_KEY_ID 替换为您创建的 KMS 密钥 ID。

   c. `aws kms put-key-policy --policy-name default --key-id $KMS_KEY_ID --policy file://path-to-kms-key-policy.json`

调用流视频事件的标签检测操作

Amazon Rekognition Video 可以检测流视频中的人员或相关物体，并在检测到时通知您。创建标签检测流处理器时，请选择您希望 Amazon Rekognition Video 检测哪些标签。这些标签可以是人员、包裹和宠物，也可以是人员、包裹、宠物。仅选择要检测的特定标签。这样，只有相关的标签才能创建通知。您可以配置选项来确定何时存储视频信息，然后根据帧中检测到的标签进行其他处理。

设置资源后，检测流视频中标签的过程如下所示：

1. 创建流处理器
2. 启动流处理器
3. 如果检测到感兴趣的物体，则当每个感兴趣的物体首次出现时，您都会收到 Amazon SNS 通知。
4. `MaxDurationInSeconds` 中指定的时间结束后，流处理器将停止。
5. 您会收到包含事件摘要的最终 Amazon SNS 通知。
6. Amazon Rekognition Video 将详细会话摘要发布到 S3 存储桶。

主题

- 创建 Amazon Rekognition Video 标签检测流处理器
- 启动 Amazon Rekognition Video 标签检测流处理器
- 分析标签检测结果

创建 Amazon Rekognition Video 标签检测流处理器

在分析流媒体视频之前，您需要先创建一个 Amazon Rekognition Video 流处理器 ()。CreateStreamProcessor

如果您想创建一个流处理器来检测感兴趣的标签和人员，请提供 Kinesis 视频流（`Input`）、Amazon S3 存储桶信息（`Output`）和 Amazon SNS 主题 ARN（`StreamProcessorNotificationChannel`）作为输入。您还可以提供 KMS 密钥 ID 来加密发送到 S3 存储桶的数据。您可以指定在 `Settings` 中要检测的内容，例如人员、包裹和人员，或宠物、人员、包裹。您还可以指定希望 Amazon Rekognition 用 `RegionsOfInterest` 监控帧中的哪个位置。以下是 `CreateStreamProcessor` 请求的 JSON 示例。

```
{
  "DataSharingPreference": { "OptIn":TRUE
  },
  "Input": {
    "KinesisVideoStream": {
      "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/muh_video_stream/
nnnnnnnnnnnn"
    }
  },
  "KmsKeyId": "muhkey",
```

```
  "Name": "muh-default_stream_processor",
  "Output": {
    "S3Destination": {
      "Bucket": "s3bucket",
      "KeyPrefix": "s3prefix"
    }
  },
  "NotificationChannel": {
    "SNSTopicArn": "arn:aws:sns:us-east-2:nnnnnnnnnnnn:MyTopic"
  },
  "RoleArn": "arn:aws:iam::nnnnnnnnn:role/Admin",
  "Settings": {
    "ConnectedHome": {
      "Labels": [
        "PET"
      ]
    "MinConfidence": 80
    }
  },
  "RegionsOfInterest": [
    {
      "BoundingBox": {
        "Top": 0.11,
        "Left": 0.22,
        "Width": 0.33,
        "Height": 0.44
      }
    },
    {
      "Polygon": [
        {
          "X": 0.11,
          "Y": 0.11
        },
        {
          "X": 0.22,
          "Y": 0.22
        },
        {
          "X": 0.33,
          "Y": 0.33
        }
      ]
    }
```

```
  ]
}
```

请注意，当您为流处理器指定 ConnectedHomeSettings 时，您可以更改 MinConfidence 值。MinConfidence 是一个介于 0 到 100 之间的数值，表示算法对其预测的确定程度。例如，置信度为 90 的person通知表示算法完全确定视频中出现了该人员。置信度值为 10 表示可能有人。您可以将 MinConfidence 值设置为介于 0 和 100 之间的所需值，具体取决于您希望收到通知的频率。例如，如果您只想在 Rekognition 绝对确定视频帧中有包裹时才收到通知，那么可以将 MinConfidence 设置为 90。

默认情况下，将 MinConfidence 设置为 50。如果要优化算法以获得更高的精确度，则可以将 MinConfidence 设置为高于 50。这样，您收到的通知就会减少，但每条通知都更加可靠。如果您想优化算法以提高查全率，则可以将 MinConfidence 设置为低于 50 以接收更多通知。

启动 Amazon Rekognition Video 标签检测流处理器

您可使用在 CreateStreamProcessor 中指定的流处理器名称来调用 StartStreamProcessor，由此开始分析流视频。在标签检测流处理器上运行 StartStreamProcessor 操作时，需要输入启动和停止信息来确定处理时间。

启动流处理器时，标签检测流处理器的状态会以下列方式发生变化：

1. 当您调用 StartStreamProcessor 时，标签检测流处理器状态从 STOPPED 或 FAILED 变为 STARTING。
2. 当标签检测流处理器运行时，它会停留在 STARTING 中。
3. 标签检测流处理器运行完毕后，状态变为 STOPPED 或 FAILED。

StartSelector 指定了 Kinesis 流中开始处理的起点。您可以使用 KVS Producer 时间戳或 KVS 片段编号。有关更多信息，请参阅片段。

> **Note**
>
> 如果您使用 KVS Producer 时间戳，则必须以毫秒为单位输入时间。

StopSelector 指定何时停止处理该流。您可以指定处理视频的最长时间。默认持续时间最长为 10 秒。请注意，实际处理时间可能会比最大持续时间长一点，具体取决于单个 KVS 片段的大小。如果片段末尾已达到或超过最大持续时间，则处理时间将停止。

以下是 StartStreamProcessor 请求的 JSON 示例。

```
{
   "Name": "string",
   "StartSelector": {
     "KVSStreamStartSelector": {
         "KVSProducerTimestamp": 1655930623123
      },
        "StopSelector": {
            "MaxDurationInSeconds": 11
      }
   }
}
```

如果流处理器成功启动，则会返回 HTTP 200 响应。包含一个空的 JSON 正文。

分析标签检测结果

Amazon Rekognition Video 可以通过三种方式发布来自标签检测流处理器的通知：针对对象检测事件的亚马逊 SNS 通知、用于摘要 end-of-session的 Amazon SNS 通知以及详细的 Amazon S3 存储桶报告。

- Amazon SNS 发送的对象检测事件通知。

  如果在视频流中检测到标签，您将收到有关对象检测事件的 Amazon SNS 通知。当在视频流中首次检测到感兴趣的物体或人员时，Amazon Rekognition 会发布通知。通知包括检测到的标签类型、置信度以及主角图片链接等信息。它们还包括被检测到的人或物体的裁剪图像和检测时间戳。通知格式如下：

  ```
  {"Subject": "Rekognition Stream Processing Event",
      "Message": {
          "inputInformation": {
              "kinesisVideo": {
                  "streamArn": string
              }
  ```

```
        },
        "eventNamespace": {
            "type": "LABEL_DETECTED"
        },
        "labels": [{
            "id": string,
            "name": "PERSON" | "PET" | "PACKAGE",
            "frameImageUri": string,
            "croppedImageUri": string,
            "videoMapping": {
                "kinesisVideoMapping": {
                    "fragmentNumber": string,
                    "serverTimestamp": number,
                    "producerTimestamp": number,
                    "frameOffsetMillis": number
                }
            },
            "boundingBox": {
                "left": number,
                "top": number,
                "height": number,
                "width": number
            }
        }],
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
    }
}
```

- 亚马逊 SNS 摘要。 end-of-session

  当流处理会话结束时，您还会收到 Amazon SNS 通知。此通知列出了会话的元数据。它包括诸如处理流的持续时间等详细信息。通知格式如下：

```
{"Subject": "Rekognition Stream Processing Event",
    "Message": {
        "inputInformation": {
```

```
            "kinesisVideo": {
                "streamArn": string,
                "processedVideoDurationMillis": number
            }
        },
        "eventNamespace": {
            "type": "STREAM_PROCESSING_COMPLETE"
        },
        "streamProcessingResults": {
            "message": string
        },
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
    }
}
```

- Amazon S3 存储桶报告。

  Amazon Rekognition Video 会将视频分析操作的详细推理结果发布到 `CreateStreamProcessor` 操作中提供的 Amazon S3 存储桶中。这些结果包括首次检测到感兴趣的物体或人员的图像帧。

  这些帧在 S3 中可用，路径如下：StreamProcessorName/ObjectKeyPrefixSessionId/*service_determined_unique_path*。在此路径中，LabelKeyPrefix是客户提供的可选参数，StreamProcessorName是流处理器资源的名称，SessionId也是流处理会话的唯一 ID。根据您的情况进行更换。

检测自定义标签

Amazon Rekognition Custom Labels 可以识别图像中特定于您的业务需求的对象和场景，如徽标或工程机器零件。有关更多信息，请参阅《Amazon Rekognition Custom Labels 开发人员指南》中的什么是 Amazon Rekognition Custom Labels？。

检测和分析人脸

Amazon Rekognition APIs 为您提供可用于检测和分析图像和视频中人脸的功能。本节概述了用于人脸分析的非存储操作。这些操作包括检测人脸标记、分析情绪和比较人脸等功能。

Amazon Rekognition 可以识别人脸标记（例如眼睛位置），检测情绪（例如开心或难过）和其他属性（例如是否戴眼镜、是否存在面部遮挡）。当检测到人脸时，系统会分析人脸属性并返回每个属性的置信度分数。

本节包含图像和视频操作的示例。

有关使用 Rekognition 的图像操作的更多信息，请参阅使用图像。

有关使用 Rekognition 的视频操作的更多信息，请参阅使用存储视频分析操作。

请注意，这些操作是非存储操作。您可以使用存储操作和人脸集合来保存图像中检测到的人脸的人脸元数据。稍后，您可以搜索图像和视频中的存储人脸。例如，这使您能够在视频中搜索特定人员。有关更多信息，请参阅 在集合中搜索人脸。

有关更多信息，请参阅亚马逊 Rek FAQs ognition 的 “面孔” 部分。

> **Note**
>
> 由 Amazon Rekognition Image 和 Amazon Rekognition Video 使用的人脸检测模型不支持检测卡通/动画人物或非人类实体中的人脸。如果您想检测图像或视频中的卡通人物，建议您使用 Amazon Rekognition Custom Labels。有关更多信息，请参阅 Amazon Rekognition Custom Labels 开发人员指南。

主题

- 人脸检测和人脸比较概述
- 人脸属性指南
- 检测图像中的人脸
- 比较图像中的人脸
- 检测存储视频中的文本

人脸检测和人脸比较概述

Amazon Rekognition 让用户能够访问两个主要的机器学习应用程序来处理包含人脸的图像，分别是人脸检测和人脸比较。它们支持人脸分析和身份验证等关键功能，因此对于从安全到个人照片整理等各种应用都至关重要。

人脸检测

人脸检测系统可以解决这样一个问题："这张图片中是否有人脸？"人脸检测的关键方面包括：

- 位置和方向：确定图像或视频帧中人脸的存在、位置、比例和方向。
- 人脸属性：无论属性如何（性别、年龄或面部毛发等），都能检测人脸。
- 其他信息：提供有关面部遮挡和眼睛凝视方向的详细信息。

人脸比较

人脸比较系统重点解决这样一个问题："一张图像中的人脸是否与另一张图像中的人脸匹配？"人脸比较系统的功能包括：

- 人脸匹配预测：将图像中的人脸与所提供数据库中的人脸进行比较，以预测匹配结果。
- 人脸属性处理：处理属性以比较人脸，无论表情、面部毛发和年龄如何。

置信度分数和漏检

人脸检测和人脸比较系统均利用置信度分数。置信度分数表示预测的可能性，例如是否存在人脸或人脸之间是否匹配。分数越高，可能性就越大。例如，90% 的置信度表示正确检测或存在匹配的可能性高于 60%。

如果人脸检测系统无法正确检测到人脸，或者对实际人脸提供低置信度预测，则为漏掉的detection/false negative. If the system incorrectly predicts the presence of a face at a high confidence level, this is a false alarm/false阳性。

同样，面部比较系统可能无法匹配属于同一个人的两张面孔（错过detection/false negative) or may incorrectly predict that two faces from different people are the same person (false alarm/false阳性）。

应用程序设计和阈值设置

- 您可以设置一个阈值来指定返回结果所需的最低置信度。选择适当的置信度阈值对于基于系统输出进行应用程序设计和决策至关重要。
- 所选的置信度应反映您的使用案例。使用案例和置信度阈值的一些示例：
- • 照片应用程序：较低阈值（例如 80%）可能足以识别照片中的家庭成员。
  - 高风险场景：在漏检或误报风险较高的使用案例（例如安全应用程序）中，系统应使用较高的置信度。在这种情况下，建议使用较高阈值（例如 99%），以实现精确的人脸匹配。

有关设置和了解置信度阈值的更多信息，请参阅在集合中搜索人脸。

人脸属性指南

以下是有关 Amazon Rekognition 如何处理和返回人脸属性的详细信息。

- FaceDetail 对象：对于每张检测到的人脸，都会返回一个 FaceDetail 对象。 FaceDetail 其中包含有关面部标志、质量、姿势等的数据。
- 属性预测：系统将预测情绪、性别、年龄等属性。系统会为每个预测分配一个置信度，并返回预测以及相应的置信度分数。对于敏感使用案例，建议使用 99% 的置信度阈值。对于年龄估算，预测年龄范围的中点提供了最佳的近似值。

请注意，性别和情绪预测基于外表，不应该用于确定实际的性别认同或情绪状态。性别二进制（男性/女性）预测基于特定图像中人脸的物理外观。它并不表示一个人的性别认同，因此您不应该使

用 Rekognition 来做出这样的决定。我们不建议使用性别二进制预测来做出会影响个人权利、隐私或服务访问权限的决策。同样，情绪预测并不表示一个人的实际内在情绪状态，因此您不应该使用 Rekognition 来做出这样的决定。在图片中假装露出笑脸的人可能看起来很快乐，但可能没有感到幸福。

应用和使用案例

以下是这些属性的一些实际应用和使用案例：

- 应用：微笑、姿势和锐度等属性可用于匿名选择个人资料照片或估算人口统计数据。
- 常见使用案例：社交媒体应用以及活动或零售店中的人口统计数据估算就是典型的例子。

有关每个属性的更多详细信息，请参阅FaceDetail。

检测图像中的人脸

Amazon Rekognition Image DetectFaces提供的操作可查找关键面部特征，例如眼睛、鼻子和嘴巴，以检测输入图像中的人脸。Amazon Rekognition Image 可检测图像中的 100 张最大人脸。

您可以提供输入图像作为图像字节数组（base64 编码的图像字节）或指定 Amazon S3 对象。在此过程中，您将图像（JPEG 或 PNG）上传到您的 S3 存储桶并指定对象键名称。

检测图像中的人脸

1. 如果您尚未执行以下操作，请：

   a. 使用 `AmazonRekognitionFullAccess` 和 `AmazonS3ReadOnlyAccess` 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将包含一张或多张人脸的图像上传到您的 S3 存储桶。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 `DetectFaces`。

Java

此示例显示检测到的人脸的估计年龄范围，并列出所有检测到的人脸属性的 JSON。将 photo 的值更改为图像文件名。将amzn-s3-demo-bucket的值更改为存储图像的 Amazon S3 存储桶。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.AgeRange;
import com.amazonaws.services.rekognition.model.Attribute;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.FaceDetail;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.List;


public class DetectFaces {


   public static void main(String[] args) throws Exception {

      String photo = "input.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      DetectFacesRequest request = new DetectFacesRequest()
         .withImage(new Image()
            .withS3Object(new S3Object()
               .withName(photo)
```

```
                .withBucket(bucket)))
            .withAttributes(Attribute.ALL);
         // Replace Attribute.ALL with Attribute.DEFAULT to get default values.

         try {
            DetectFacesResult result = rekognitionClient.detectFaces(request);
            List < FaceDetail > faceDetails = result.getFaceDetails();

            for (FaceDetail face: faceDetails) {
               if (request.getAttributes().contains("ALL")) {
                  AgeRange ageRange = face.getAgeRange();
                  System.out.println("The detected face is estimated to be between
 "
                     + ageRange.getLow().toString() + " and " +
 ageRange.getHigh().toString()
                     + " years old.");
                  System.out.println("Here's the complete set of attributes:");
               } else { // non-default attributes have null values.
                  System.out.println("Here's the default set of attributes:");
               }

               ObjectMapper objectMapper = new ObjectMapper();

 System.out.println(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(fac
            }

         } catch (AmazonRekognitionException e) {
            e.printStackTrace();
         }

      }

}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import java.util.List;

//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.AgeRange;

//snippet-end:[rekognition.java2.detect_labels.import]

public class DetectFaces {

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,amzn-s3-demo-bucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
```

```
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

        try {
            S3Object s3Object = S3Object.builder()
                .bucket(bucket)
                .name(image)
                .build() ;

            Image myImage = Image.builder()
                .s3Object(s3Object)
                .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                    .attributes(Attribute.ALL)
                    .image(myImage)
                    .build();

                DetectFacesResponse facesResponse =
 rekClient.detectFaces(facesRequest);
                List<FaceDetail> faceDetails = facesResponse.faceDetails();
                for (FaceDetail face : faceDetails) {
                    AgeRange ageRange = face.ageRange();
                    System.out.println("The detected face is estimated to be
 between "
                                + ageRange.low().toString() + " and " +
 ageRange.high().toString()
                                + " years old.");

                    System.out.println("There is a smile :
 "+face.smile().value().toString());
                }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

AWS CLI

此示例显示detect-faces AWS CLI 操作的 JSON 输出。将 file 替换为图像文件的名称。将amzn-s3-demo-bucket替换为包含图像文件的 Amazon S3 存储桶的名称。

```
aws rekognition detect-faces --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket,"Name":"image-name"}}"\
                              --attributes "ALL" --profile profile-name --region
 region-name
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition detect-faces --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --attributes "ALL"
--profile profile-name --region region-name
```

Python

此示例显示检测到的人脸的估计年龄范围和其他属性，并列出所有检测到的人脸属性的 JSON。将 photo 的值更改为图像文件名。将amzn-s3-demo-bucket的值更改为存储图像的 Amazon S3 存储桶。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
import boto3
import json

def detect_faces(photo, bucket, region):

    session = boto3.Session(profile_name='profile-name',
                            region_name=region)
    client = session.client('rekognition', region_name=region)

    response = client.detect_faces(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
                                   Attributes=['ALL'])
```

```
    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']
['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Face Occluded: " + str(faceDetail['FaceOccluded']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

def main():
    photo='photo'
    bucket='amzn-s3-demo-bucket'
    region='region'
    face_count=detect_faces(photo, bucket, region)
    print("Faces detected: " + str(face_count))

if __name__ == "__main__":
    main()
```

.NET

此示例显示检测到的人脸的估计年龄范围，并列出所有检测到的人脸属性的 JSON。将 photo 的值更改为图像文件名。将amzn-s3-demo-bucket的值更改为存储图像的 Amazon S3 存储桶。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
public class DetectFaces
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectFacesRequest detectFacesRequest = new DetectFacesRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            // Attributes can be "ALL" or "DEFAULT".
            // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
            // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
items/Rekognition/TFaceDetail.html
            Attributes = new List<String>() { "ALL" }
        };

        try
        {
            DetectFacesResponse detectFacesResponse =
 rekognitionClient.DetectFaces(detectFacesRequest);
            bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
            foreach(FaceDetail face in detectFacesResponse.FaceDetails)
            {
                Console.WriteLine("BoundingBox: top={0} left={1} width={2}
 height={3}", face.BoundingBox.Left,
                    face.BoundingBox.Top, face.BoundingBox.Width,
 face.BoundingBox.Height);
                Console.WriteLine("Confidence: {0}\nLandmarks: {1}\nPose:
 pitch={2} roll={3} yaw={4}\nQuality: {5}",
                    face.Confidence, face.Landmarks.Count, face.Pose.Pitch,
                    face.Pose.Roll, face.Pose.Yaw, face.Quality);
                if (hasAll)
```

```
                    Console.WriteLine("The detected face is estimated to be
 between " +
                        face.AgeRange.Low + " and " + face.AgeRange.High + "
 years old.");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Ruby

此示例显示检测到的人脸的估计年龄范围，并列出各种人脸属性。将 photo 的值更改为图像文件名。将amzn-s3-demo-bucket的值更改为存储图像的 Amazon S3 存储桶。

```
  # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket = 'bucket' # the bucketname without s3://
  photo  = 'input.jpg'# the name of file
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    image: {
      s3_object: {
        bucket: bucket,
        name: photo
      },
    },
    attributes: ['ALL']
  }
  response = client.detect_faces attrs
  puts "Detected faces for: #{photo}"
  response.face_details.each do |face_detail|
    low  = face_detail.age_range.low
```

```
    high = face_detail.age_range.high
    puts "The detected face is between: #{low} and #{high} years old"
    puts "All other attributes:"
    puts "  bounding_box.width:     #{face_detail.bounding_box.width}"
    puts "  bounding_box.height:    #{face_detail.bounding_box.height}"
    puts "  bounding_box.left:      #{face_detail.bounding_box.left}"
    puts "  bounding_box.top:       #{face_detail.bounding_box.top}"
    puts "  age.range.low:          #{face_detail.age_range.low}"
    puts "  age.range.high:         #{face_detail.age_range.high}"
    puts "  smile.value:            #{face_detail.smile.value}"
    puts "  smile.confidence:       #{face_detail.smile.confidence}"
    puts "  eyeglasses.value:       #{face_detail.eyeglasses.value}"
    puts "  eyeglasses.confidence:  #{face_detail.eyeglasses.confidence}"
    puts "  sunglasses.value:       #{face_detail.sunglasses.value}"
    puts "  sunglasses.confidence:  #{face_detail.sunglasses.confidence}"
    puts "  gender.value:           #{face_detail.gender.value}"
    puts "  gender.confidence:      #{face_detail.gender.confidence}"
    puts "  beard.value:            #{face_detail.beard.value}"
    puts "  beard.confidence:       #{face_detail.beard.confidence}"
    puts "  mustache.value:         #{face_detail.mustache.value}"
    puts "  mustache.confidence:    #{face_detail.mustache.confidence}"
    puts "  eyes_open.value:        #{face_detail.eyes_open.value}"
    puts "  eyes_open.confidence:   #{face_detail.eyes_open.confidence}"
    puts "  mout_open.value:        #{face_detail.mouth_open.value}"
    puts "  mout_open.confidence:   #{face_detail.mouth_open.confidence}"
    puts "  emotions[0].type:       #{face_detail.emotions[0].type}"
    puts "  emotions[0].confidence: #{face_detail.emotions[0].confidence}"
    puts "  landmarks[0].type:      #{face_detail.landmarks[0].type}"
    puts "  landmarks[0].x:         #{face_detail.landmarks[0].x}"
    puts "  landmarks[0].y:         #{face_detail.landmarks[0].y}"
    puts "  pose.roll:              #{face_detail.pose.roll}"
    puts "  pose.yaw:               #{face_detail.pose.yaw}"
    puts "  pose.pitch:             #{face_detail.pose.pitch}"
    puts "  quality.brightness:     #{face_detail.quality.brightness}"
    puts "  quality.sharpness:      #{face_detail.quality.sharpness}"
    puts "  confidence:             #{face_detail.confidence}"
    puts "------------"
    puts ""
  end
```

Node.js

此示例显示检测到的人脸的估计年龄范围，并列出各种人脸属性。将 photo 的值更改为图像文件名。将amzn-s3-demo-bucket的值更改为存储图像的 Amazon S3 存储桶。

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

如果您使用的是 TypeScript 定义，则可能需要使用import AWS from 'aws-sdk'而不是const AWS = require('aws-sdk')，以便使用 Node.js 运行该程序。您可以查阅适用于 JavaScript 的AWS SDK，了解更多详情。根据您的配置设置方式，您可能还需要使用AWS.config.update({region:*region*});来指定您的区域。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucketname without s3://
const photo  = 'photo-name' // the name of file

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
  Attributes: ['ALL']
}

client.detectFaces(params, function(err, response) {
    if (err) {
      console.log(err, err.stack); // an error occurred
    } else {
      console.log(`Detected faces for: ${photo}`)
      response.FaceDetails.forEach(data => {
        let low  = data.AgeRange.Low
        let high = data.AgeRange.High
```

```
        console.log(`The detected face is between: ${low} and ${high} years
old`)
        console.log("All other attributes:")
        console.log(`  BoundingBox.Width:      ${data.BoundingBox.Width}`)
        console.log(`  BoundingBox.Height:     ${data.BoundingBox.Height}`)
        console.log(`  BoundingBox.Left:       ${data.BoundingBox.Left}`)
        console.log(`  BoundingBox.Top:        ${data.BoundingBox.Top}`)
        console.log(`  Age.Range.Low:          ${data.AgeRange.Low}`)
        console.log(`  Age.Range.High:         ${data.AgeRange.High}`)
        console.log(`  Smile.Value:            ${data.Smile.Value}`)
        console.log(`  Smile.Confidence:       ${data.Smile.Confidence}`)
        console.log(`  Eyeglasses.Value:       ${data.Eyeglasses.Value}`)
        console.log(`  Eyeglasses.Confidence:  ${data.Eyeglasses.Confidence}`)
        console.log(`  Sunglasses.Value:       ${data.Sunglasses.Value}`)
        console.log(`  Sunglasses.Confidence:  ${data.Sunglasses.Confidence}`)
        console.log(`  Gender.Value:           ${data.Gender.Value}`)
        console.log(`  Gender.Confidence:      ${data.Gender.Confidence}`)
        console.log(`  Beard.Value:            ${data.Beard.Value}`)
        console.log(`  Beard.Confidence:       ${data.Beard.Confidence}`)
        console.log(`  Mustache.Value:         ${data.Mustache.Value}`)
        console.log(`  Mustache.Confidence:    ${data.Mustache.Confidence}`)
        console.log(`  EyesOpen.Value:         ${data.EyesOpen.Value}`)
        console.log(`  EyesOpen.Confidence:    ${data.EyesOpen.Confidence}`)
        console.log(`  MouthOpen.Value:        ${data.MouthOpen.Value}`)
        console.log(`  MouthOpen.Confidence:   ${data.MouthOpen.Confidence}`)
        console.log(`  Emotions[0].Type:       ${data.Emotions[0].Type}`)
        console.log(`  Emotions[0].Confidence: ${data.Emotions[0].Confidence}`)
        console.log(`  Landmarks[0].Type:      ${data.Landmarks[0].Type}`)
        console.log(`  Landmarks[0].X:         ${data.Landmarks[0].X}`)
        console.log(`  Landmarks[0].Y:         ${data.Landmarks[0].Y}`)
        console.log(`  Pose.Roll:              ${data.Pose.Roll}`)
        console.log(`  Pose.Yaw:               ${data.Pose.Yaw}`)
        console.log(`  Pose.Pitch:             ${data.Pose.Pitch}`)
        console.log(`  Quality.Brightness:     ${data.Quality.Brightness}`)
        console.log(`  Quality.Sharpness:      ${data.Quality.Sharpness}`)
        console.log(`  Confidence:             ${data.Confidence}`)
        console.log("------------")
        console.log("")
      }) // for response.faceDetails
    } // if
  });
```

DetectFaces 操作请求

对 DetectFaces 的输入是一个图像。在此示例中，图像从 Amazon S3 存储桶加载。Attributes 参数指定应返回所有人脸属性。有关更多信息，请参阅 使用图像。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "Attributes": [
        "ALL"
    ]
}
```

DetectFaces 操作响应

DetectFaces 将返回每个检测到的人脸的以下信息：

- 边界框 – 人脸周围的边界框的坐标。
- 置信度 - 边界框包含人脸的置信度级别。
- 人脸标记 - 一组人脸标记。对于每个标记（例如，左眼、右眼和嘴），此响应将提供 x 坐标和 y 坐标。
- 面部属性 - 一组面部属性，例如脸部是否被遮挡，作为 FaceDetail 对象返回。该套装包括：AgeRange、Beard、Emotions、 EyeDirection、Eyeglasses、、 EyesOpen、Gend FaceOccluded er、、Mustache MouthOpen、Smile 和太阳镜。对于每个此类属性，此响应将提供一个值。该值可以是不同的类型，例如布尔值类型（人员是否戴了太阳镜）或字符串(人员是男性还是女性），或角度值（眼睛凝视方向的俯仰/偏转）。此外，对于大多数属性，此响应还为属性提供检测到的值的置信度。请注意，虽然使用时支持 FaceOccluded 和 EyeDirection 属性DetectFaces，但使用StartFaceDetection和分析视频时不支持这些属性GetFaceDetection。
- 质量 - 描述人脸的亮度和锐度。有关确保尽可能最佳的人脸检测的信息，请参阅有关面部比较输入图像的建议。
- 姿势 - 描述图像中的人脸的旋转。

该请求可以描绘您想要返回的面部属性数组。将始终返回面部属性的DEFAULT子集 - BoundingBox、Confidence、Pose、Quality 和 Landmarks。您可以使用["DEFAULT", "FACE_OCCLUDED", "EYE_DIRECTION"]或只使用一个属性（比如["FACE_OCCLUDED"]）来请求返回特定的面部属性（除了默认列表之外）。您可以使用["ALL"]请求所有面部属性。请求更多属性可能会增加响应时间。

下面是 DetectFaces API 调用的示例响应：


```
{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Width": 0.7919622659683228,
        "Height": 0.7510867118835449,
        "Left": 0.08881539851427078,
        "Top": 0.151064932346344
      },
      "AgeRange": {
        "Low": 18,
        "High": 26
      },
      "Smile": {
        "Value": false,
        "Confidence": 89.77348327636719
      },
      "Eyeglasses": {
        "Value": true,
        "Confidence": 99.99996948242188
      },
      "Sunglasses": {
        "Value": true,
        "Confidence": 93.65237426757812
      },
      "Gender": {
        "Value": "Female",
        "Confidence": 99.85968780517578
      },
      "Beard": {
        "Value": false,
        "Confidence": 77.52591705322266
      },
      "Mustache": {
        "Value": false,
```

```
        "Confidence": 94.48904418945312
      },
      "EyesOpen": {
        "Value": true,
        "Confidence": 98.57169342041016
      },
      "MouthOpen": {
        "Value": false,
        "Confidence": 74.33953094482422
      },
      "Emotions": [
        {
          "Type": "SAD",
          "Confidence": 65.56403350830078
        },
        {
          "Type": "CONFUSED",
          "Confidence": 31.277774810791016
        },
        {
          "Type": "DISGUSTED",
          "Confidence": 15.553778648376465
        },
        {
          "Type": "ANGRY",
          "Confidence": 8.012762069702148
        },
        {
          "Type": "SURPRISED",
          "Confidence": 7.621500015258789
        },
        {
          "Type": "FEAR",
          "Confidence": 7.243380546569824
        },
        {
          "Type": "CALM",
          "Confidence": 5.8196024894714355
        },
        {
          "Type": "HAPPY",
          "Confidence": 2.2830512523651123
        }
      ],
```

```
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "eyeRight",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "mouthLeft",
          "X": 0.343580037355423,
          "Y": 0.6951127648353577
        },
        {
          "Type": "mouthRight",
          "X": 0.6306480765342712,
          "Y": 0.6898072361946106
        },
        {
          "Type": "nose",
          "X": 0.47164231538772583,
          "Y": 0.5763645172119141
        },
        {
          "Type": "leftEyeBrowLeft",
          "X": 0.1732882857322693,
          "Y": 0.34452149271965027
        },
        {
          "Type": "leftEyeBrowRight",
          "X": 0.3655243515968323,
          "Y": 0.33231860399246216
        },
        {
          "Type": "leftEyeBrowUp",
          "X": 0.2671719491481781,
          "Y": 0.31669262051582336
        },
        {
          "Type": "rightEyeBrowLeft",
          "X": 0.5613729953765869,
```

```
      "Y": 0.32813435792922974
    },
    {
      "Type": "rightEyeBrowRight",
      "X": 0.7665090560913086,
      "Y": 0.3318614959716797
    },
    {
      "Type": "rightEyeBrowUp",
      "X": 0.6612788438796997,
      "Y": 0.3082450032234192
    },
    {
      "Type": "leftEyeLeft",
      "X": 0.2416982799768448,
      "Y": 0.4085965156555176
    },
    {
      "Type": "leftEyeRight",
      "X": 0.36943578720092773,
      "Y": 0.41230902075767517
    },
    {
      "Type": "leftEyeUp",
      "X": 0.29974061250686646,
      "Y": 0.3971870541572571
    },
    {
      "Type": "leftEyeDown",
      "X": 0.30360740423202515,
      "Y": 0.42347756028175354
    },
    {
      "Type": "rightEyeLeft",
      "X": 0.5755768418312073,
      "Y": 0.4081145226955414
    },
    {
      "Type": "rightEyeRight",
      "X": 0.7050536870956421,
      "Y": 0.39924031496047974
    },
    {
      "Type": "rightEyeUp",
```

```
          "X": 0.642906129360199,
          "Y": 0.39026668667793274
        },
        {
          "Type": "rightEyeDown",
          "X": 0.6423097848892212,
          "Y": 0.41669243574142456
        },
        {
          "Type": "noseLeft",
          "X": 0.4122826159000397,
          "Y": 0.5987403392791748
        },
        {
          "Type": "noseRight",
          "X": 0.5394935011863708,
          "Y": 0.5960900187492371
        },
        {
          "Type": "mouthUp",
          "X": 0.478581964969635,
          "Y": 0.6660456657409668
        },
        {
          "Type": "mouthDown",
          "X": 0.483366996049881,
          "Y": 0.7497162818908691
        },
        {
          "Type": "leftPupil",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "rightPupil",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "upperJawlineLeft",
          "X": 0.11031254380941391,
          "Y": 0.3980775475502014
        },
        {
```

```
          "Type": "midJawlineLeft",
          "X": 0.19301874935626984,
          "Y": 0.7034031748771667
        },
        {
          "Type": "chinBottom",
          "X": 0.4939905107021332,
          "Y": 0.8877836465835571
        },
        {
          "Type": "midJawlineRight",
          "X": 0.7990140914916992,
          "Y": 0.6899225115776062
        },
        {
          "Type": "upperJawlineRight",
          "X": 0.8548634648323059,
          "Y": 0.38160091638565063
        }
      ],
      "Pose": {
        "Roll": -5.83309268951416,
        "Yaw": -2.4244730472564697,
        "Pitch": 2.6216139793395996
      },
      "Quality": {
        "Brightness": 96.16363525390625,
        "Sharpness": 95.51618957519531
      },
      "Confidence": 99.99872589111328,
      "FaceOccluded": {
        "Value": true,
        "Confidence": 99.99726104736328
      },
      "EyeDirection": {
        "Yaw": 16.299732,
        "Pitch": -6.407457,
        "Confidence": 99.968704
      }
    }
  ],
  "ResponseMetadata": {
    "RequestId": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
    "HTTPStatusCode": 200,
```

```
    "HTTPHeaders": {
      "x-amzn-requestid": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "3409",
      "date": "Wed, 26 Apr 2023 20:18:50 GMT"
    },
    "RetryAttempts": 0
  }
}
```

请注意以下几点：

- Pose 数据描述了检测到的人脸的旋转。您可以使用 BoundingBox 和 Pose 数据的组合，在应用程序显示的人脸的四周绘制边界框。
- Quality 描述人脸的亮度和锐度。您可能会发现这对比较图像之间的人脸和查找最佳人脸会很有用。
- 前面的响应显示服务可检测的所有人脸 landmarks、所有人脸属性和情绪。要在响应中获取所有这些项，您必须指定 attributes 参数与值 ALL。默认情况下，DetectFaces API 仅返回以下 5 个人脸属性：BoundingBox、Confidence、Pose、Quality 和 landmarks。返回的默认标记为：eyeLeft、eyeRight、nose、mouthLeft 和 mouthRight。

比较图像中的人脸

使用 Rekognition，您可以使用该操作比较两张图像之间的面孔。CompareFaces此特征对于身份验证或照片匹配等应用非常有用。

CompareFaces 将源图像中的人脸与目标图像中的每张人脸进行比较。图像通过以下 CompareFaces 任一方式传递给：

- 图像的 Base64 编码表示。
- Amazon S3 对象。

人脸检测与人脸比较

人脸比较不同于人脸检测。人脸检测（使用 DetectFaces）仅识别图像或视频中人脸的存在和位置。相比之下，人脸比较涉及将源图像中检测到的人脸与目标图像中的人脸进行比较以找到匹配项。

相似度阈值

使用 similarityThreshold 参数定义响应中要包含的匹配项的最低置信度。默认情况下，响应中只返回相似度得分大于或等于 80% 的人脸。

> **Note**
>
> CompareFaces 使用机器学习算法，这些算法是概率性的。假阴性是一种错误的预测，即与源图像中的人脸相比，目标图像中的人脸具有较低的相似性置信度得分。为了降低假阴性的可能性，我们建议您将目标图像与多个源图像进行比较。如果您计划使用 CompareFaces 来做出影响个人权利、隐私或服务访问权限的决定，我们建议您在采取行动之前将结果交给人类进行审查和进一步验证。

以下代码示例演示了如何使用各种 CompareFaces 操作 AWS SDKs。在 AWS CLI 示例中，您将两张 JPEG 图像上传到您的 Amazon S3 存储桶并指定对象密钥名称。在其他示例中，您从本地文件系统中加载两个文件并将它们作为图像字节数组输入。

比较人脸

1. 如果您尚未执行以下操作，请：
   a. 创建或更新具有AmazonRekognitionFullAccess和AmazonS3ReadOnlyAccess（仅限 AWS CLI 示例）权限的用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下示例代码调用 CompareFaces 操作。

   Java

   此示例显示有关与从本地文件系统加载的源和目标图像中的人脸进行匹配的信息。

   将 sourceImage 和 targetImage 的值分别替换为源和目标图像的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
```

```
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.CompareFacesMatch;
import com.amazonaws.services.rekognition.model.CompareFacesRequest;
import com.amazonaws.services.rekognition.model.CompareFacesResult;
import com.amazonaws.services.rekognition.model.ComparedFace;
import java.util.List;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;

public class CompareFaces {

   public static void main(String[] args) throws Exception{
       Float similarityThreshold = 70F;
       String sourceImage = "source.jpg";
       String targetImage = "target.jpg";
       ByteBuffer sourceImageBytes=null;
       ByteBuffer targetImageBytes=null;

       AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

       //Load source and target images and create input parameters
       try (InputStream inputStream = new FileInputStream(new
 File(sourceImage))) {
          sourceImageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
       }
       catch(Exception e)
       {
           System.out.println("Failed to load source image " + sourceImage);
           System.exit(1);
       }
       try (InputStream inputStream = new FileInputStream(new
 File(targetImage))) {
           targetImageBytes =
 ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
       }
       catch(Exception e)
       {
           System.out.println("Failed to load target images: " + targetImage);
           System.exit(1);
```

```
        }

        Image source=new Image()
             .withBytes(sourceImageBytes);
        Image target=new Image()
             .withBytes(targetImageBytes);

        CompareFacesRequest request = new CompareFacesRequest()
                .withSourceImage(source)
                .withTargetImage(target)
                .withSimilarityThreshold(similarityThreshold);

        // Call operation
        CompareFacesResult
 compareFacesResult=rekognitionClient.compareFaces(request);


        // Display results
        List <CompareFacesMatch> faceDetails =
 compareFacesResult.getFaceMatches();
        for (CompareFacesMatch match: faceDetails){
          ComparedFace face= match.getFace();
          BoundingBox position = face.getBoundingBox();
          System.out.println("Face at " + position.getLeft().toString()
                + " " + position.getTop()
                + " matches with " + match.getSimilarity().toString()
                + "% confidence.");

        }
        List<ComparedFace> uncompared = compareFacesResult.getUnmatchedFaces();

        System.out.println("There was " + uncompared.size()
             + " face(s) that did not match");
    }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import java.util.List;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.CompareFacesMatch;
import software.amazon.awssdk.services.rekognition.model.CompareFacesRequest;
import software.amazon.awssdk.services.rekognition.model.CompareFacesResponse;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

// snippet-end:[rekognition.java2.detect_faces.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CompareFaces {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <pathSource> <pathTarget>\n\n" +
            "Where:\n" +
            "   pathSource - The path to the source image (for example, C:\\AWS\
\pic1.png). \n " +
            "   pathTarget - The path to the target image (for example, C:\\AWS\
\pic2.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        Float similarityThreshold = 70F;
```

```
        String sourceImage = args[0];
        String targetImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        compareTwoFaces(rekClient, similarityThreshold, sourceImage,
 targetImage);
        rekClient.close();
   }

    // snippet-start:[rekognition.java2.compare_faces.main]
    public static void compareTwoFaces(RekognitionClient rekClient, Float
 similarityThreshold, String sourceImage, String targetImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            InputStream tarStream = new FileInputStream(targetImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            SdkBytes targetBytes = SdkBytes.fromInputStream(tarStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            Image tarImage = Image.builder()
                .bytes(targetBytes)
                .build();

            CompareFacesRequest facesRequest = CompareFacesRequest.builder()
                .sourceImage(souImage)
                .targetImage(tarImage)
                .similarityThreshold(similarityThreshold)
                .build();

            // Compare the two images.
            CompareFacesResponse compareFacesResult =
 rekClient.compareFaces(facesRequest);
            List<CompareFacesMatch> faceDetails =
 compareFacesResult.faceMatches();
            for (CompareFacesMatch match: faceDetails){
```

```
                ComparedFace face= match.face();
                BoundingBox position = face.boundingBox();
                System.out.println("Face at " + position.left().toString()
                        + " " + position.top()
                        + " matches with " + face.confidence().toString()
                        + "% confidence.");

            }
            List<ComparedFace> uncompared = compareFacesResult.unmatchedFaces();
            System.out.println("There was " + uncompared.size() + " face(s) that
 did not match");
            System.out.println("Source image rotation: " +
 compareFacesResult.sourceImageOrientationCorrection());
            System.out.println("target image rotation: " +
 compareFacesResult.targetImageOrientationCorrection());

        } catch(RekognitionException | FileNotFoundException e) {
            System.out.println("Failed to load source image " + sourceImage);
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.compare_faces.main]
}
```

AWS CLI

此示例显示compare-faces AWS CLI 操作的 JSON 输出。

将amzn-s3-demo-bucket替换为包含源和目标图像的 Amazon S3 存储桶的名称。将 source.jpg 和 target.jpg 替换为源和目标图像的文件名。

```
aws rekognition compare-faces --target-image \
"{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" \
--source-image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-
name"}}"
--profile profile-name
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition compare-faces --target-image "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--source-image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":
\"image-name\"}}" --profile profile-name
```

Python

此示例显示有关与从本地文件系统加载的源和目标图像中的人脸进行匹配的信息。

将 source_file 和 target_file 的值分别替换为源和目标图像的路径和文件名。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def compare_faces(sourceFile, targetFile):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    imageSource = open(sourceFile, 'rb')
    imageTarget = open(targetFile, 'rb')

    response = client.compare_faces(SimilarityThreshold=80,
                                    SourceImage={'Bytes': imageSource.read()},
                                    TargetImage={'Bytes': imageTarget.read()})

    for faceMatch in response['FaceMatches']:
        position = faceMatch['Face']['BoundingBox']
        similarity = str(faceMatch['Similarity'])
        print('The face at ' +
              str(position['Left']) + ' ' +
              str(position['Top']) +
              ' matches with ' + similarity + '% confidence')

    imageSource.close()
    imageTarget.close()
    return len(response['FaceMatches'])
```

```
def main():
    source_file = 'source-file-name'
    target_file = 'target-file-name'
    face_matches = compare_faces(source_file, target_file)
    print("Face matches: " + str(face_matches))

if __name__ == "__main__":
    main()
```

.NET

此示例显示有关与从本地文件系统加载的源和目标图像中的人脸进行匹配的信息。

将 sourceImage 和 targetImage 的值分别替换为源和目标图像的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CompareFaces
{
    public static void Example()
    {
        float similarityThreshold = 70F;
        String sourceImage = "source.jpg";
        String targetImage = "target.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        Amazon.Rekognition.Model.Image imageSource = new
 Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(sourceImage, FileMode.Open,
 FileAccess.Read))
            {
```

```
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load source image: " + sourceImage);
            return;
        }

        Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read))
            {
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load target image: " + targetImage);
            return;
        }

        CompareFacesRequest compareFacesRequest = new CompareFacesRequest()
        {
            SourceImage = imageSource,
            TargetImage = imageTarget,
            SimilarityThreshold = similarityThreshold
        };

        // Call operation
        CompareFacesResponse compareFacesResponse =
rekognitionClient.CompareFaces(compareFacesRequest);

        // Display results
        foreach(CompareFacesMatch match in compareFacesResponse.FaceMatches)
        {
```

```
            ComparedFace face = match.Face;
            BoundingBox position = face.BoundingBox;
            Console.WriteLine("Face at " + position.Left
                  + " " + position.Top
                  + " matches with " + match.Similarity
                  + "% confidence.");
        }

        Console.WriteLine("There was " +
 compareFacesResponse.UnmatchedFaces.Count + " face(s) that did not match");

    }
}
```

Ruby

此示例显示有关与从本地文件系统加载的源和目标图像中的人脸进行匹配的信息。

将 photo_source 和 photo_target 的值分别替换为源和目标图像的路径和文件名。

```
 # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket        = 'bucket' # the bucketname without s3://
  photo_source  = 'source.jpg'
  photo_target  = 'target.jpg'
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    source_image: {
      s3_object: {
        bucket: bucket,
        name: photo_source
      },
    },
    target_image: {
      s3_object: {
        bucket: bucket,
        name: photo_target
      },
```

```
    },
    similarity_threshold: 70
  }
  response = client.compare_faces attrs
  response.face_matches.each do |face_match|
    position   = face_match.face.bounding_box
    similarity = face_match.similarity
    puts "The face at: #{position.left}, #{position.top} matches with
 #{similarity} % confidence"
  end
```

Node.js

此示例显示有关与从本地文件系统加载的源和目标图像中的人脸进行匹配的信息。

将 photo_source 和 photo_target 的值分别替换为源和目标图像的路径和文件名。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucket name without s3://
const photo_source  = 'photo-source-name' // path and the name of file
const photo_target = 'photo-target-name'

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

const client = new AWS.Rekognition();
   const params = {
     SourceImage: {
       S3Object: {
         Bucket: bucket,
         Name: photo_source
       },
     },
     TargetImage: {
       S3Object: {
         Bucket: bucket,
         Name: photo_target
       },
     },
     SimilarityThreshold: 70
```

```
  }
  client.compareFaces(params, function(err, response) {
    if (err) {
      console.log(err, err.stack); // an error occurred
    } else {
      response.FaceMatches.forEach(data => {
        let position   = data.Face.BoundingBox
        let similarity = data.Similarity
        console.log(`The face at: ${position.Left}, ${position.Top} matches
with ${similarity} % confidence`)
      }) // for response.faceDetails
    } // if
  });
```

CompareFaces 操作请求

对 CompareFaces 的输入是一个图像。在本示例中，源和目标图像是从本地文件系统加载的。SimilarityThreshold 输入参数指定所比较的人脸必须匹配以包含在响应中的最小置信度。有关更多信息，请参阅 使用图像。

```
{
    "SourceImage": {
        "Bytes": "/9j/4AAQSk2Q==..."
    },
    "TargetImage": {
        "Bytes": "/9j/4O1Q==..."
    },
    "SimilarityThreshold": 70
}
```

CompareFaces 操作响应

响应包括以下内容：

- 一组匹配的人脸：包含每张匹配人脸的相似度得分和元数据的匹配人脸列表。如果多张人脸匹配，则 faceMatches

  数组包括所有人脸匹配。
- 人脸匹配详细信息：每张匹配的人脸还提供边界框、置信度值、标记位置和相似度得分。

- 不匹配的人脸列表：响应还包括目标图像中与源图像人脸不匹配的人脸。包括每张不匹配人脸的边界框。
- 源人脸信息：包含有关用于比较的源图像中的人脸的信息（包括边界框和置信度值）。

该示例显示已在目标图像中找到一个人脸匹配。对于该人脸匹配，它提供了一个边界框和一个置信度值（Amazon Rekognition 在边界框中包含人脸的置信度值）。相似度得分为 99.99 指示人脸的相似程度。该示例也显示 Amazon Rekognition 在目标图像中找到的与源图像中分析的人脸不匹配的一张人脸。


```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
```

```
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.26942989230155945,
                    "X": 0.6454493403434753,
                    "Type": "mouthRight"
                }
            ]
        },
        "Similarity": 100.0
    }],
    "SourceImageOrientationCorrection": "ROTATE_90",
    "TargetImageOrientationCorrection": "ROTATE_90",
    "UnmatchedFaces": [{
        "BoundingBox": {
            "Width": 0.4890109896659851,
            "Top": 0.6566604375839233,
            "Left": 0.10989011079072952,
            "Height": 0.278298944234848
        },
        "Confidence": 99.99992370605469,
        "Pose": {
            "Yaw": 51.51519012451172,
            "Roll": -110.32493591308594,
            "Pitch": -2.322134017944336
        },
        "Quality": {
            "Sharpness": 99.99671173095703,
            "Brightness": 57.23163986206055
        },
        "Landmarks": [{
                "Y": 0.8288310766220093,
                "X": 0.3133862614631653,
                "Type": "eyeLeft"
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
```

```
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.4856561422348025,
                "Type": "mouthLeft"
            },
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```


检测存储视频中的文本

Amazon Rekognition Video 可以在 Amazon S3 存储桶存储的视频中检测人脸并提供一些信息，例如：

- 在视频中检测到人脸的次数。
- 人脸被检测到时在视频帧中的位置。
- 人脸标记，例如左眼的位置。
- 其他属性如the section called “人脸属性指南”页面所述。

存储视频中的 Amazon Rekognition Video 人脸检测是一个异步操作。要开始检测视频中的人脸，请致电StartFaceDetection。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple

Notification Service (Amazon SNS) 主题。如果视频分析成功，您可以调用 GetFaceDetection 来获取视频分析结果。有关启动视频分析和获取结果的详细信息，请参阅调用 Amazon Rekognition Video 操作。

此过程在使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon Simple Queue Service (Amazon SQS) 队列获取视频分析请求的完成状态）中的代码的基础上进行了扩展。

检测存储在 Amazon S3 存储桶内的视频中的人脸 (SDK)

1. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
2. 将以下代码添加到您在步骤 1 中创建的类 `VideoDetect`。

   AWS CLI

   - 在以下代码示例中，将`amzn-s3-demo-bucket`和`video-name`更改为您在步骤 2 中指定的 Amazon S3 存储桶名称和文件名。
   - 将`region-name`更改为您使用的 AWS 区域。将`profile_name`的值替换为您的开发人员资料的名称。
   - 将 `TopicARN` 更改为您在 配置 Amazon Rekognition Video 的步骤 3 中创建的 Amazon SNS 主题的 ARN。
   - 将 `RoleARN` 更改为您在 配置 Amazon Rekognition Video 的步骤 7 中创建的 IAM 服务角色的 ARN。

```
aws rekognition start-face-detection --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"Video-Name"}}" --notification-channel
"{"SNSTopicArn":"Topic-ARN","RoleArn":"Role-ARN"}" --region region-name --
profile profile-name
```

   如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition start-face-detection --video "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"Video-Name\"}}" --notification-channel
"{\"SNSTopicArn\":\"Topic-ARN\",\"RoleArn\":\"Role-ARN\"}" --region region-name
 --profile profile-name
```

运行 StartFaceDetection 操作并获取任务 ID 号后，运行以下 GetFaceDetection 操作并提供任务 ID 号：

```
aws rekognition get-face-detection --job-id job-id-number  --profile profile-
name
```

Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


private static void StartFaceDetection(String bucket, String video) throws
 Exception{

    NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);

    StartFaceDetectionRequest req = new StartFaceDetectionRequest()
            .withVideo(new Video()
                    .withS3Object(new S3Object()
                        .withBucket(bucket)
                        .withName(video)))
            .withNotificationChannel(channel);



    StartFaceDetectionResult startLabelDetectionResult =
 rek.startFaceDetection(req);
    startJobId=startLabelDetectionResult.getJobId();

}

private static void GetFaceDetectionResults() throws Exception{

    int maxResults=10;
```

```
    String paginationToken=null;
    GetFaceDetectionResult faceDetectionResult=null;

    do{
        if (faceDetectionResult !=null){
            paginationToken = faceDetectionResult.getNextToken();
        }

        faceDetectionResult = rek.getFaceDetection(new
 GetFaceDetectionRequest()
             .withJobId(startJobId)
             .withNextToken(paginationToken)
             .withMaxResults(maxResults));

        VideoMetadata videoMetaData=faceDetectionResult.getVideoMetadata();

        System.out.println("Format: " + videoMetaData.getFormat());
        System.out.println("Codec: " + videoMetaData.getCodec());
        System.out.println("Duration: " + videoMetaData.getDurationMillis());
        System.out.println("FrameRate: " + videoMetaData.getFrameRate());


        //Show faces, confidence and detection times
        List<FaceDetection> faces= faceDetectionResult.getFaces();

        for (FaceDetection face: faces) {
            long seconds=face.getTimestamp()/1000;
            System.out.print("Sec: " + Long.toString(seconds) + " ");
            System.out.println(face.getFace().toString());
            System.out.println();
        }
    } while (faceDetectionResult !=null && faceDetectionResult.getNextToken() !=
 null);


}
```

在函数 main 中，将以下行:

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

替换为:

```
        StartFaceDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetFaceDetectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.recognize_video_faces.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_faces.import]


/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class VideoDetectFaces {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
         "   video - The name of video (for example, people.mp4). \n\n" +
```

```
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    StartFaceDetection(rekClient, channel, bucket, video);
    GetFaceResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_faces.main]
public static void StartFaceDetection(RekognitionClient rekClient,
                                       NotificationChannel channel,
                                       String bucket,
                                       String video) {

    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();
```

```
        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartFaceDetectionRequest  faceDetectionRequest =
StartFaceDetectionRequest.builder()
            .jobTag("Faces")
            .faceAttributes(FaceAttributes.ALL)
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartFaceDetectionResponse startLabelDetectionResult =
rekClient.startFaceDetection(faceDetectionRequest);
        startJobId=startLabelDetectionResult.jobId();

    } catch(RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetFaceResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetFaceDetectionResponse faceDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (faceDetectionResponse !=null)
                paginationToken = faceDetectionResponse.nextToken();

            GetFaceDetectionRequest recognitionRequest =
GetFaceDetectionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds
            while (!finished) {
```

```
                faceDetectionResponse =
rekClient.getFaceDetection(recognitionRequest);
                status = faceDetectionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null
            VideoMetadata videoMetaData=faceDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            // Show face information
            List<FaceDetection> faces= faceDetectionResponse.faces();

            for (FaceDetection face: faces) {
                String age = face.face().ageRange().toString();
                String smile = face.face().smile().toString();
                System.out.println("The detected face is estimated to be"
                            + age + " years old.");
                System.out.println("There is a smile : "+smile);
            }

        } while (faceDetectionResponse !=null &&
faceDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.recognize_video_faces.main]
```

```
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Faces===============
    def StartFaceDetection(self):
        response=self.rek.start_face_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetFaceDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_face_detection(JobId=self.startJobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for faceDetection in response['Faces']:
                print('Face: ' + str(faceDetection['Face']))
                print('Confidence: ' + str(faceDetection['Face']['Confidence']))
                print('Timestamp: ' + str(faceDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
```

```
            else:
                finished = True
```

在函数 `main` 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    analyzer.StartFaceDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceDetectionResults()
```

> **Note**
>
> 如果您已运行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)之外的视频示例，则要替换的函数名称将不同。

3. 运行该代码。将显示有关在视频中检测到的人脸的信息。

GetFaceDetection 操作响应

`GetFaceDetection` 将返回一个数组 (`Faces`)，其中包含有关在视频中检测到的人脸的信息。每当在视频中检测到人脸时，都会存在一个数组元素 FaceDetection。返回的数组元素按时间顺序排列，从视频开始起计时，以毫秒为单位。

以下示例是来自 `GetFaceDetection` 的部分 JSON 响应。在响应中，请注意以下内容：

- 边界框 – 人脸周围的边界框的坐标。
- 置信度 - 边界框包含人脸的置信度级别。
- 人脸标记 - 一组人脸标记。对于每个标记（例如，左眼、右眼和嘴），此响应将提供 x 坐标和 y 坐标。
- 面部属性 — 一组面部属性，包括： AgeRange、胡子、情绪、眼镜、性别、 EyesOpen、胡子 MouthOpen、微笑和太阳镜。该值可以是不同的类型，例如布尔值类型（人员是否戴了太阳镜）或字符串（人员是男性还是女性）。此外，对于大多数属性，此响应还为属性提供检测到的值

的置信度。请注意，虽然使用时支持 FaceOccluded 和 EyeDirection属性DetectFaces，但使用StartFaceDetection和分析视频时不支持这些属性GetFaceDetection。

- 时间戳 - 视频中检测到人脸的时间。
- 分页信息 - 此示例显示一页人脸检测信息。您可以在 GetFaceDetection 的 MaxResults 输入参数中指定要返回的人员元素数量。如果存在的结果的数量超过了 MaxResults，则 GetFaceDetection 会返回一个令牌 (NextToken)，用于获取下一页的结果。有关更多信息，请参阅 获取 Amazon Rekognition Video 分析结果。
- 视频信息 - 此响应包含有关由 VideoMetadata 返回的每页信息中的视频格式 (GetFaceDetection) 的信息。
- 质量 - 描述人脸的亮度和锐度。
- 姿势 - 描述人脸的旋转。


```
{
    "Faces": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
                "Confidence": 99.97504425048828,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.46415066719055176,
                        "Y": 0.2572723925113678
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.5068183541297913,
                        "Y": 0.23705792427062988
                    },
                    {
                        "Type": "nose",
                        "X": 0.49765899777412415,
                        "Y": 0.28383663296699524
                    },
```

```
                    {
                        "Type": "mouthLeft",
                        "X": 0.487221896648407,
                        "Y": 0.3452930748462677
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.5142884850502014,
                        "Y": 0.33167609572410583
                    }
                ],
                "Pose": {
                    "Pitch": 15.966927528381348,
                    "Roll": -15.547388076782227,
                    "Yaw": 11.34195613861084
                },
                "Quality": {
                    "Brightness": 44.80223083496094,
                    "Sharpness": 99.95819854736328
                }
            },
            "Timestamp": 0
        },
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.20000000298023224,
                    "Left": 0.029999999329447746,
                    "Top": 0.2199999988079071,
                    "Width": 0.11249999701976776
                },
                "Confidence": 99.85971069335938,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.06842322647571564,
                        "Y": 0.3010137975215912
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.10543643683195114,
                        "Y": 0.29697132110595703
                    },
                    {
```

```
                    "Type": "nose",
                    "X": 0.09569807350635529,
                    "Y": 0.33701086044311523
                },
                {
                    "Type": "mouthLeft",
                    "X": 0.0732642263174057,
                    "Y": 0.3757539987564087
                },
                {
                    "Type": "mouthRight",
                    "X": 0.10589495301246643,
                    "Y": 0.3722417950630188
                }
            ],
            "Pose": {
                "Pitch": -0.5589138865470886,
                "Roll": -5.1093974113464355,
                "Yaw": 18.69594955444336
            },
            "Quality": {
                "Brightness": 43.052337646484375,
                "Sharpness": 99.68138885498047
            }
        },
        "Timestamp": 0
    },
    {
        "Face": {
            "BoundingBox": {
                "Height": 0.2177777737379074,
                "Left": 0.7593749761581421,
                "Top": 0.13333334028720856,
                "Width": 0.12250000238418579
            },
            "Confidence": 99.63436889648438,
            "Landmarks": [
                {
                    "Type": "eyeLeft",
                    "X": 0.8005779385566711,
                    "Y": 0.20915353298187256
                },
                {
                    "Type": "eyeRight",
```

```
                        "X": 0.8391435146331787,
                        "Y": 0.21049551665782928
                    },
                    {
                        "Type": "nose",
                        "X": 0.8191410899162292,
                        "Y": 0.2523227035999298
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.8093273043632507,
                        "Y": 0.29053622484207153
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.8366993069648743,
                        "Y": 0.29101791977882385
                    }
                ],
                "Pose": {
                    "Pitch": 3.165884017944336,
                    "Roll": 1.4182015657424927,
                    "Yaw": -11.151537895202637
                },
                "Quality": {
                    "Brightness": 28.910892486572266,
                    "Sharpness": 97.61507415771484
                }
            },
            "Timestamp": 0
        }.......

    ],
    "JobStatus": "SUCCEEDED",
    "NextToken": "i7fj5XPV/
fwviXqz0eag9Ow332Jd5G8ZGWf7hooirD/6V1qFmjKFOQZ6QPWUiqv29HbyuhMNqQ==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
```

```
    }
}
```

在集合中搜索人脸

Amazon Rekognition 可让您使用输入人脸在存储人脸集合中搜索匹配的人脸。首先，将检测到的人脸信息存储在称为“集合”的服务器端容器中。集合存储单张人脸和单个用户（同一个人有几张人脸）。单张人脸以人脸向量的形式存储，人脸向量是人脸的数学表示（不是人脸的实际图像）。同一个人的不同图像可用于创建多个人脸向量并存储在同一集合中。然后，您可以汇总同一个人的多个人脸向量来创建用户向量。用户向量可以提供更高的人脸搜索精确度，并提供更可靠的描绘，包括不同程度的光照、锐度、姿势、外观等。

创建集合后，您可以使用输入面容在集合中搜索匹配的用户向量或人脸向量。与针对单个人脸向量进行搜索相比，根据用户向量进行搜索可以显著提高准确性。您可以使用在图像、存储视频和流视频中检测到的人脸来搜索存储的人脸向量。您可以使用在图像中检测到的人脸来搜索存储的用户向量。

要存储人脸信息，您需要执行以下操作：

1. 创建收藏夹-要存储面部信息，您必须先在账户中的一个 AWS 区域中创建 (CreateCollection) 面部集合。在调用 `IndexFaces` 操作时指定此人脸集合。
2. 人脸索引-该IndexFaces操作可检测图像中的人脸，提取人脸矢量并将其存储在集合中。您可使用此操作检测图像中的人脸，并将有关检测到的人脸特征的信息保存到集合中。这是基于存储的 API 操作示例，因为服务会将人脸向量信息存储在服务器上。

要创建用户并将多个人脸向量与用户关联，您需要执行以下操作：

1. 创建用户-必须先使用创建用户CreateUser。您可以通过将同一个人的多个人脸向量汇总到一个用户向量中来提高人脸匹配的准确性。您最多可以将 100 个人脸向量与一个用户向量相关联。
2. 关联面孔-创建用户后，您可以通过AssociateFaces操作向该用户添加现有人脸矢量。人脸向量必须与用户向量位于同一个集合中，才能与该用户向量相关联。

创建集合并存储人脸和用户向量后，您可以使用以下操作来搜索匹配的人脸：

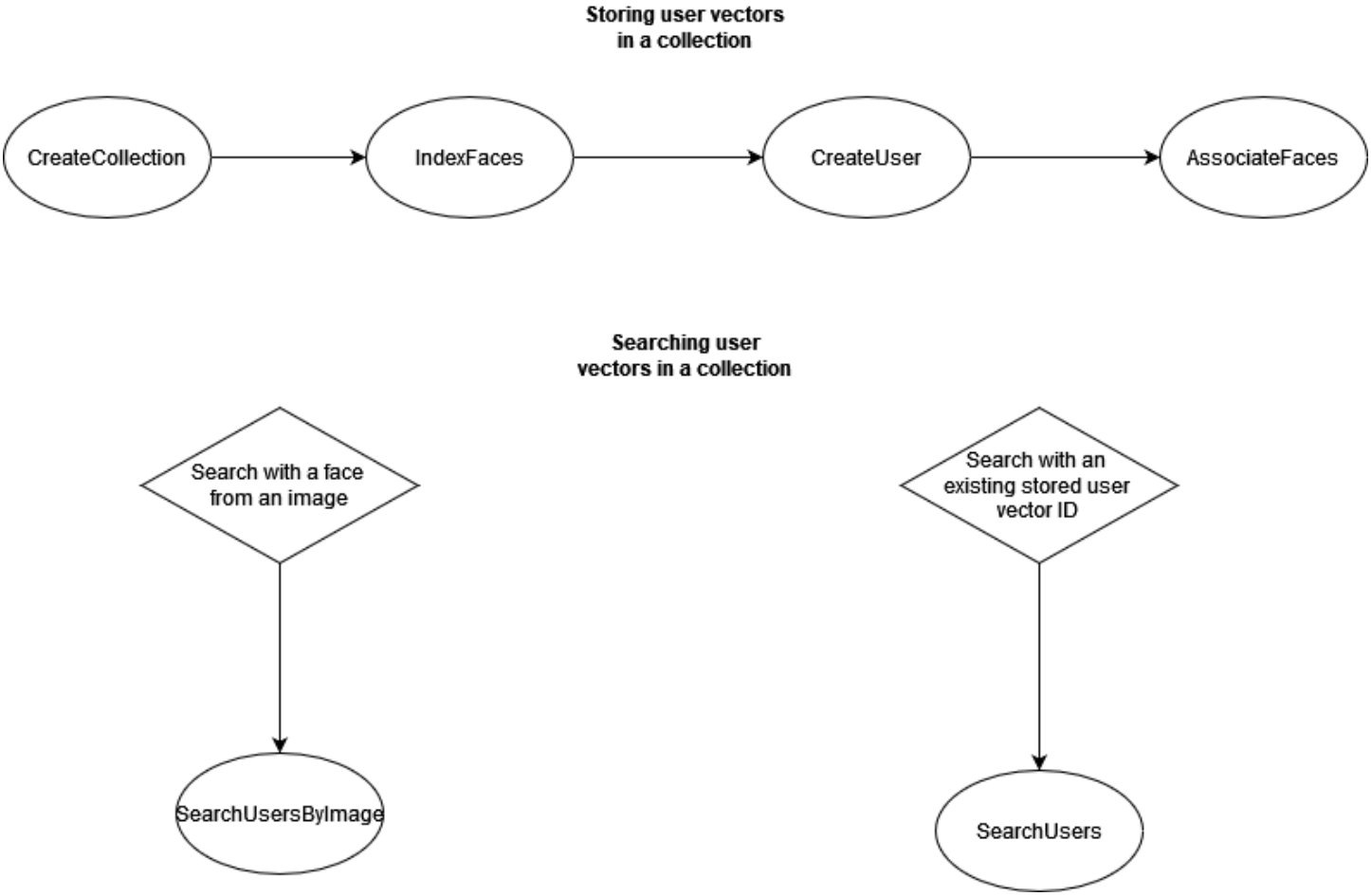
- SearchFacesByImage-使用图像中的人脸搜索存储的单个面孔。
- SearchFaces-使用提供的面容 ID 搜索存储的个人面孔。
- SearchUsers-使用提供的面容 ID 或用户 ID 对存储的用户进行搜索。
- SearchUsersByImage-使用图像中的面孔搜索存储的用户。
- StartFaceSearch-在存储的视频中搜索人脸。

- CreateStreamProcessor-在直播视频中搜索面孔。

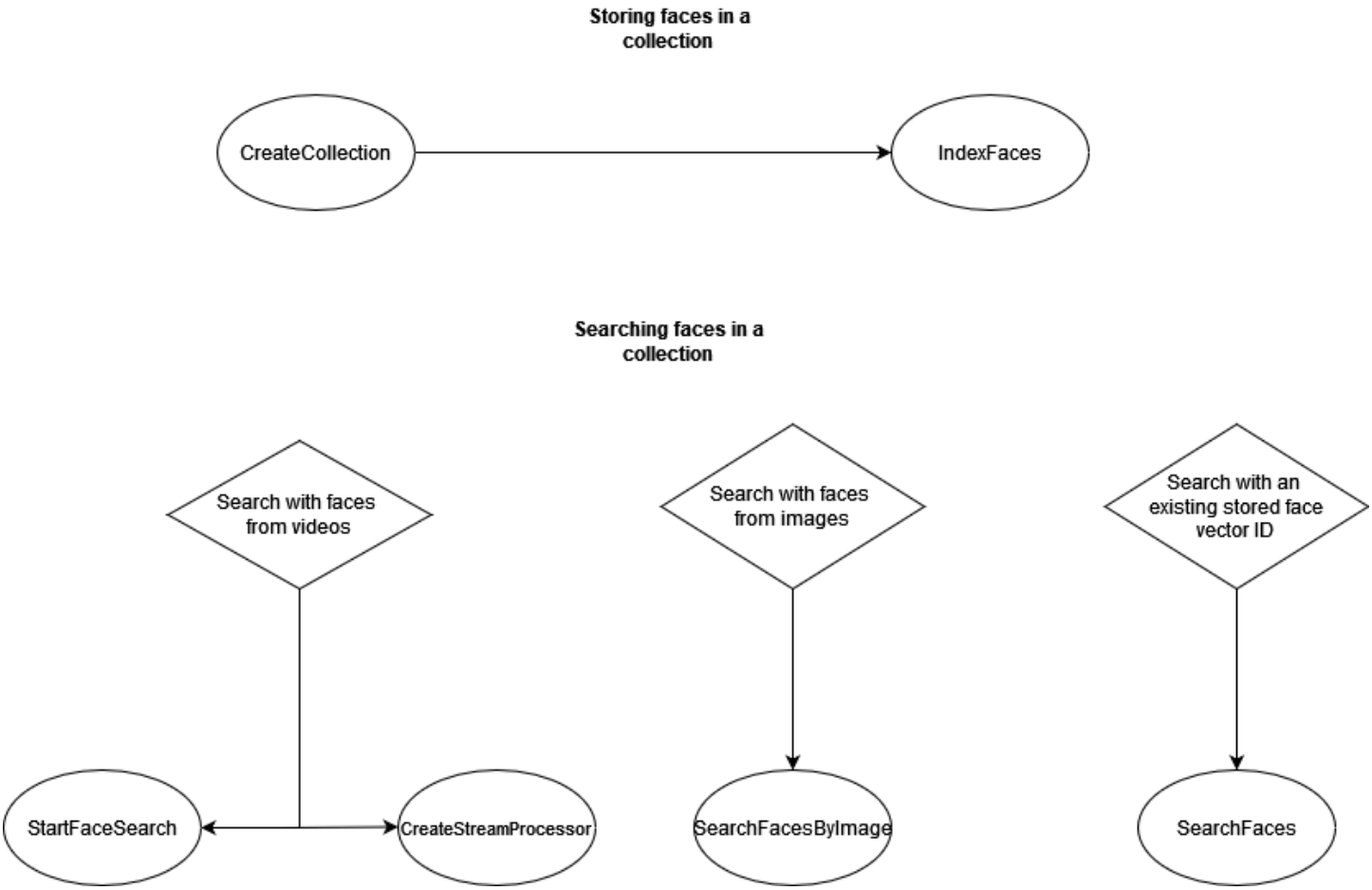
> **Note**
>
> 集合存储人脸向量，这些向量是人脸的数学表示。集合不存储人脸图像。

下图根据您使用集合的目标显示了调用操作的顺序：

为了更大限度地提高与用户向量匹配的精度：



为了与单个人脸向量进行高精度匹配：



在许多情况下都可以使用集合。例如，您可以使用和AssociateFaces操作创建一个人脸集合，用于存储从扫描的员工徽章图像中检测到的人脸和 IDs 政府签发的人脸。IndexFaces当员工进入大楼时，将捕获员工的人脸图像，并将此图像发送到 SearchUsersByImage 操作。如果人脸匹配得到了足够高的相似度得分（例如 99%），则可对员工进行身份验证。

管理人脸集合、人脸和用户

人脸集合是 Amazon Rekognition 用来包含与人脸和用户相关信息的资源。您可以使用各种 Rekognition API 操作来管理集合本身，以及存储在集合中的任何人脸或人脸向量。

管理集合

人脸集合是主要 Amazon Rekognition 资源，而您创建的每个人脸集合均具有一个唯一的 Amazon 资源名称 (ARN)。你可以在账户的特定 AWS 区域创建每个面孔集合。创建集合时，它会与人脸检测模型的最新版本关联。有关更多信息，请参阅 了解模型版本控制。

您可以对集合执行以下管理操作：

- 使用 CreateCollection 创建集合。有关更多信息，请参阅 创建集合。
- 使用 ListCollections 列出可用的集合。有关更多信息，请参阅 列出集合。
- 使用 DescribeCollection 描述集合。有关更多信息，请参阅 描述集合。
- 使用 DeleteCollection 删除集合。有关更多信息，请参阅 删除集合。

管理集合中的人脸

在创建人脸集合后，您可以将人脸存储到该集合中。Amazon Rekognition 提供以下操作来管理集合中的人脸：

- IndexFaces 操作可检测输入图像 (JPEG 或 PNG) 中的人脸，并将这些人脸添加到指定的人脸集合。将为在图像中检测到的每个人脸返回一个唯一的人脸 ID。在保留人脸后，您可以在人脸集合中搜索匹配的人脸。有关更多信息，请参阅 将人脸添加到集合。
- ListFaces 操作列出集合中的人脸。有关更多信息，请参阅 将人脸添加到集合。
- DeleteFaces 操作删除集合中的人脸。有关更多信息，请参阅 从集合中删除人脸。

在集合中管理用户

存储来自同一个人的多个人脸向量后，您可以通过将所有这些人脸向量关联到一个用户向量中来提高准确性。您可以使用以下操作来管理用户：

- CreateUser-Operation 使用提供的唯一用户 ID 在集合中创建新用户。
- AssociateUsers-在用户 ID 中添加 1-100 张唯一面孔 IDs 。将至少一个人脸 ID 与用户关联后，您可以在您的集合中搜索与该用户的匹配项。
- ListUsers-列出集合中的用户。
- DeleteUsers-使用提供的用户 ID 从集合中删除用户。
- DisassociateFaces-移除用户的一张或 IDs 多张面孔。

使用相似度阈值关联和匹配人脸

相似度阈值用于关联和匹配人脸。以下是针对这两个使用案例使用相似度阈值的指南。

使用相似度阈值关联人脸

使用AssociateFaces操作关联人脸时，重要的是要确保与用户关联的人脸全部来自同一个人。为提供帮助，UserMatchThreshold 参数指定了新面孔与已包含至少一个 FaceID 的 UserID 关联所需的最低用户匹配置信度。这有助于确保 FaceIds 与正确的 UserID 相关联。该值的范围为 0-100，默认值为 75。

使用相似性阈值匹配人脸

我们允许您通过提供相似度阈值作为输入参数来控制所有搜索操作的结果（CompareFacesSearchFacesSearchFacesByImageSearchUsers、、、、SearchUsersByImage）。

FaceMatchThreshold 是 SearchFaces 和 SearchFacesByImage 的相似性阈值输入属性，它根据与被匹配人脸的相似性来控制返回的结果数。SearchUsers 和 SearchUsersByImage 的相似性阈值属性是 UserMatchThreshold，它根据与被匹配用户向量的相似性来控制返回的结果数。阈值属性为 CompareFaces 的 SimilarityThreshold。

对于响应，如果其 Similarity 响应属性值低于阈值，则不返回。此阈值对于校准使用案例非常重要，因为它可以确定匹配结果中包含的误报数。此阈值控制搜索结果的查全率；该阈值越低，查全率越高。

所有机器学习系统是概率性的。您应根据自己的使用案例，判断如何设置合适的相似性阈值。例如，如果您打算生成一个照片应用来标识长相相似的家庭成员，则可选择较低的阈值（例如，80%）。另一方面，对于许多执法使用案例，我们建议您使用 99% 或更高的阈值来减少意外错误。

除了 FaceMatchThreshold 和 UserMatchThreshold，还可以使用 Similarity 响应属性来减少意外错误。例如，您可以选择使用较低的阈值（如 80%）以返回更多结果。然后，您可以使用响应属性相似性（相似性的百分比）来缩小选择范围以及在应用程序中筛选合适的响应。再次强调，使用越高的相似性（例如 99% 及以上）越能降低识别出错的风险。

常见场景中的人脸索引指南

下面是在常见情景中使用 IndexFaces 的指南。

关键或公共安全应用

- 使用仅包含每个图像中的一个人脸且将返回的人脸 ID 与图像主体的标识符关联的图像调用 IndexFaces。

- 您可以在索引之前使用 DetectFaces 验证图像中是否只有一个人脸。如果检测到多个人脸，则在审查后重新提交图像并且只让一个人脸呈现。这将防止无意中索引多个人脸并将它们与同一个人关联。

照片共享和社交媒体应用

- 当对使用案例中包含多个人脸的图像（如家庭相册）没有任何限制时，您应该调用 IndexFaces。在这种情况下，您需要识别每张照片中的每个人，并使用该信息根据照片中出现的人对照片进行分组。

一般使用

- 对同一个人的多个不同图像进行索引，尤其是对具有不同人脸属性（人脸姿态、面部毛发等）的图像进行索引、创建用户、关联针对该用户的不同人脸，以提高匹配质量。
- 增加一个审查过程，使您可以用正确的人脸标识符为失败的匹配编制索引，以提高后续的人脸匹配能力。
- 有关图像质量的信息，请参阅有关面部比较输入图像的建议。

搜索集合内的人脸和用户

在创建人脸集合并存储人脸向量和/或用户向量后，您可以在人脸集合中搜索匹配的人脸。使用 Amazon Rekognition，您可以搜索集合中与以下条件匹配的人脸：

- 所提供的人脸 ID (SearchFaces)。有关更多信息，请参阅 使用人脸 ID 搜索人脸。
- 所提供图像中的最大人脸 (SearchFacesByImage)。有关更多信息，请参阅 使用图像搜索人脸。
- 存储视频中的人脸。有关更多信息，请参阅 搜索存储视频中的人脸。
- 流视频中的人脸。有关更多信息，请参阅 使用流视频事件。

您可以使用 CompareFaces 操作将源图像中的人脸与目标图像中的人脸进行比较。此比较的范围限制为在目标图像中检测到的人脸。有关更多信息，请参阅比较图像中的人脸。

以下列表中显示的各种搜索操作将人脸（由 FaceId 或输入图像识别）与存储在给定人脸集合中的所有人脸进行比较：

- SearchFaces
- SearchFacesByImage

- SearchUsers
- SearchUsersByImage

涉及公共安全的使用案例

在涉及公共安全的使用案例中部署人脸检测和比较系统时，除了传感器、输入图像和视频的最佳实践和常见场景中的人脸索引指南中列出的建议之外，您还应遵循以下最佳实践。首先，您应使用 99% 或更高的置信度阈值来减少错误和误报。其次，您应增加人工审查者来验证从人脸检测或比较系统收到的结果，并且不应在没有增加人工审查的情况下根据系统输出做出决策。人脸检测和比较系统应作为一种工具，帮助缩小范围，并允许人们快速审查和考虑选择。再次，我们建议您让这些使用案例中的人脸检测和比较系统的使用公开透明，这包括尽可能向最终用户和当事人告知对这些系统的使用，获得此类使用的同意以及提供一个供最终用户和主体提供反馈以改进系统的机制。

如果您是执法机构，要在刑事调查中使用 Amazon Rekognition 人脸比较功能，则必须遵守 AWS 服务条款中列出的要求。这包括以下这些内容。

- 由经过适当培训的人员审查所有决定，以采取可能侵犯个人的公民自由或同等人权的行动。
- 培训人员如何负责任地使用人脸识别系统。
- 公开披露您使用人脸识别系统的情况。
- 在没有独立审查或迫切需要的情况下，不要使用 Amazon Rekognition 持续监视人员。

在所有情况下，人脸比较匹配均应在其他有力证据的背景下考虑，而不应该用作采取行动的唯一决定因素。但是，如果在 non-law-enforcement场景中使用面部比较（例如，解锁手机或验证员工的身份以进入安全的私人办公楼），则这些决定不需要人工审计，因为它们不会影响个人的公民自由或同等人权。

如果您计划在涉及公共安全的使用案例中使用人脸检测或人脸比较系统，则应采用前面所述的最佳实践。此外，您还应参考已发布的关于人脸比较使用的资源。这包括司法部的司法援助局提供的在刑事情报和调查活动中使用的人脸识别政策制定模板。该模板提供了一些人脸比较和生物比较相关资源，旨在为执法和公共安全机构提供一个框架，用于制定符合适用法律的人脸比较政策、降低隐私风险和建立实体责任制和监督制。其他资源包括美国国家电信与信息管理局提供的人脸识别商用最佳隐私实践和美国联邦贸易委员会工作人员提供的人脸识别一般用途最佳实践。将来可能会开发和发布其他资源，您应不断地自学此重要主题。

需要注意的是，在使用 AWS 服务时，您必须遵守所有适用的法律，并且您不得以侵犯他人权利或可能对他人有害的方式使用任何 AWS 服务。这意味着您不得以非法歧视他人或侵犯他人的正当程序、隐私或公民自由的方式将 AWS 服务用于公共安全使用案例。必要时，您应获得适当的法律建议以审查关于您的使用案例的任何法律要求或问题。

使用 Amazon Rekognition 来帮助公共安全

Amazon Rekognition 可为公共安全和执法机构使用场景提供帮助，例如寻找丢失的儿童、打击人口走私或者预防犯罪。在公共安全和执法机构使用场景中，请考虑以下事项：

- 使用 Amazon Rekognition 作为查找可能的匹配项的第一步。通过 Amazon Rekognition 人脸操作的响应，您可以快速获取一组可能的匹配以进行进一步的调查。
- 对于需要人工分析的场景，请勿使用 Amazon Rekognition 响应自主进行决策。如果您代表执法机构使用 Amazon Rekognition 协助查明与刑事调查有关的人员，并且将根据识别结果采取可能侵犯个人的公民权利或同等人权，则采取行动的决定必须由受过适当培训的人员根据他们对识别证据的独立审查作出。
- 在必须确保高度准确的人脸相似性匹配的场景，使用 99% 的相似性阈值。例如，进入大楼的身份验证。
- 当涉及到公民权利时，例如涉及执法的使用案例，请使用 99% 或更高的置信度阈值，并对人脸比较预测进行人工审查，以确保人员的公民权利不会受到侵犯。
- 对于可以获益于筛选大量可能匹配的场景，使用低于 99% 的相似性阈值。例如查找失踪人员。如有必要，您可以使用相似性响应属性来确定对于您要识别的人员，可能的匹配相似度有多高。
- 对于由 Amazon Rekognition 返回的结果，需要制定针对面部匹配误报的计划。例如，使用 IndexFaces 操作构建索引时，使用同一个人的多个图像可以改进匹配。有关更多信息，请参阅 常见场景中的人脸索引指南。

在其他使用案例（如社交媒体）中，我们建议您运用最佳判断力来评测 Amazon Rekognition 结果是否需要人工审查。此外，根据您应用程序的需求，相似性阈值可以较低。

创建集合

您可以使用 CreateCollection 操作创建集合。

有关更多信息，请参阅 管理集合。

创建集合 (开发工具包)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 CreateCollection 操作。

Java

以下示例创建一个集合，并显示其 Amazon 资源名称 (ARN)。

将 collectionId 的值更改为您要创建的集合的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateCollectionRequest;
import com.amazonaws.services.rekognition.model.CreateCollectionResult;


public class CreateCollection {

   public static void main(String[] args) throws Exception {


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      String collectionId = "MyCollection";
            System.out.println("Creating collection: " +
         collectionId );

        CreateCollectionRequest request = new CreateCollectionRequest()
                    .withCollectionId(collectionId);

      CreateCollectionResult createCollectionResult =
 rekognitionClient.createCollection(request);
      System.out.println("CollectionArn : " +
         createCollectionResult.getCollectionArn());
      System.out.println("Status code : " +
```

```
            createCollectionResult.getStatusCode().toString());

    }

}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
//snippet-start:[rekognition.java2.create_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.create_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class CreateCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName> \n\n" +
         "Where:\n" +
         "   collectionName - The name of the collection. \n\n";
```

```
    if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String collectionId = args[0];
    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    System.out.println("Creating collection: " +collectionId);
    createMyCollection(rekClient, collectionId );
    rekClient.close();
}

// snippet-start:[rekognition.java2.create_collection.main]
public static void createMyCollection(RekognitionClient rekClient,String
collectionId ) {

    try {
        CreateCollectionRequest collectionRequest =
CreateCollectionRequest.builder()
            .collectionId(collectionId)
            .build();

        CreateCollectionResponse collectionResponse =
rekClient.createCollection(collectionRequest);
        System.out.println("CollectionArn: " +
collectionResponse.collectionArn());
        System.out.println("Status code: " +
collectionResponse.statusCode().toString());

    } catch(RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.create_collection.main]
```

AWS CLI

此 AWS CLI 命令显示 create-collection CLI 操作的 JSON 输出。

将 collection-id 的值替换为您要创建的集合的名称。

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition create-collection --profile profile-name --collection-id
 "collection-name"
```

Python

以下示例创建一个集合，并显示其 Amazon 资源名称 (ARN)。

将 collection_id 的值更改为您要创建的集合的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def create_collection(collection_id):
    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    # Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId=collection_id)
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

def main():
    collection_id = "collection-id"
    create_collection(collection_id)

if __name__ == "__main__":
    main()
```

.NET

以下示例创建一个集合，并显示其 Amazon 资源名称 (ARN)。

将 collectionId 的值更改为您要创建的集合的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CreateCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        String collectionId = "MyCollection";
        Console.WriteLine("Creating collection: " + collectionId);

        CreateCollectionRequest createCollectionRequest = new
 CreateCollectionRequest()
        {
            CollectionId = collectionId
        };

        CreateCollectionResponse createCollectionResponse =
 rekognitionClient.CreateCollection(createCollectionRequest);
        Console.WriteLine("CollectionArn : " +
 createCollectionResponse.CollectionArn);
        Console.WriteLine("Status code : " +
 createCollectionResponse.StatusCode);

    }
}
```

Node.JS

在以下示例中，将region的值替换为与您的账户关联的区域名称，并将 collectionName 的值替换为所需的集合名称。

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import { CreateCollectionCommand} from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const collectionName = "collection-name"
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

const createCollection = async (collectionName) => {
    try {
       console.log(`Creating collection: ${collectionName}`)
       const data = await rekogClient.send(new
 CreateCollectionCommand({CollectionId: collectionName}));
       console.log("Collection ARN:")
       console.log(data.CollectionARN)
       console.log("Status Code:")
       console.log(String(data.StatusCode))
       console.log("Success.",  data);
       return data;
    } catch (err) {
      console.log("Error", err.stack);
    }
  };

 createCollection(collectionName)
```

CreateCollection 操作请求

`CreationCollection` 的输入是您要创建的集合的名称。

```
{
    "CollectionId": "MyCollection"
}
```

CreateCollection 操作响应

Amazon Rekognition 将创建集合，并返回最新创建的集合的 Amazon 资源名称 (ARN)。

```
{
   "CollectionArn": "aws:rekognition:us-east-1:acct-id:collection/examplecollection",
   "StatusCode": 200
}
```

标记集合

您可以使用标签识别、整理、搜索和筛选 Amazon Rekognition 集合。每个标签都是由用户定义的键和值组成的标签。

您还可以使用 Identity and Access Management (IAM) 使用标签控制集合的访问权限。有关更多信息，请参阅使用 AWS 资源标签控制对资源的访问权限。

主题

- 向新集合添加标签
- 向现有集合添加标签
- 列出集合中的标签
- 从集合中删除标签

向新集合添加标签

您可以使用 `CreateCollection` 操作在创建集合时向其添加标签。请在 `Tags` 数组输入参数中指定一个或多个标签。

AWS CLI

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition create-collection --collection-id "collection-name" --tags
 "{"key1":"value1","key2":"value2"}" --profile profile-name
```

对于 Windows 设备：

```
aws rekognition create-collection --collection-id "collection-name" --tags
 "{\"key1\":\"value1\",\"key2\":\"value2\"}" --profile profile-name
```

Python

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
import boto3

def create_collection(collection_id):
    client = boto3.client('rekognition')

    # Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId=collection_id)
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

def main():
    collection_id = 'NewCollectionName'
    create_collection(collection_id)

if __name__ == "__main__":
    main()
```

向现有集合添加标签

要将一个或多个标签添加到现有集合，请使用 TagResource 操作。指定集合的 Amazon 资源名称（ARN）（ResourceArn）和要添加的标签（Tags）。以下示例说明了如何添加两个标签。

AWS CLI

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition tag-resource --resource-arn collection-arn --tags
 "{"key1":"value1","key2":"value2"}" --profile profile-name
```

对于 Windows 设备：

```
aws rekognition tag-resource --resource-arn collection-arn --tags "{\"key1\":
\"value1\",\"key2\":\"value2\"}" --profile profile-name
```

Python

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def create_tag(collection_id):
    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')
    response = client.tag_resource(ResourceArn=collection_id,
                                   Tags={
                                       "KeyName": "ValueName"
                                   })
    print(response)
    if "'HTTPStatusCode': 200" in str(response):
        print("Success!!")

def main():
    collection_arn = "collection-arn"
    create_tag(collection_arn)

if __name__ == "__main__":
    main()
```

> **Note**
>
> 如果您不知道该集合的 Amazon 资源名称，则可以使用 DescribeCollection 操作。

列出集合中的标签

要列出附加到集合的标签，请使用 ListTagsForResource 操作并指定该集合的 ARN（ResourceArn）。该操作的响应是附加到指定集合的标签键和值的映射图。

AWS CLI

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition list-tags-for-resource --resource-arn resource-arn --profile
 profile-name
```

Python

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
import boto3

def list_tags():
    client = boto3.client('rekognition')
    response =
 client.list_tags_for_resource(ResourceArn="arn:aws:rekognition:region-
name:5498347593847598:collection/NewCollectionName")
    print(response)

def main():
    list_tags()

if __name__ == "__main__":
    main()
```

输出将显示附加到集合的标签列表：

```
                {
```

```
    "Tags": {
        "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
    }
}
```

从集合中删除标签

要从集合中删除一个或多个标签，请使用 UntagResource 操作。指定模型的 ARN（ResourceArn）和要移除的标签键（Tag-Keys）。

AWS CLI

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition untag-resource --resource-arn resource-arn --profile profile-name --
tag-keys "key1" "key2"
```

或者，您可以按以下格式指定标签键：

```
--tag-keys key1,key2
```

Python

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
import boto3

def list_tags():
    client = boto3.client('rekognition')
    response = client.untag_resource(ResourceArn="arn:aws:rekognition:region-
name:5498347593847598:collection/NewCollectionName", TagKeys=['KeyName'])
    print(response)

def main():
    list_tags()

if __name__ == "__main__":
    main()
```

列出集合

您可以使用 ListCollections 操作列出您正在使用的区域中的集合。

有关更多信息，请参阅 管理集合。

列出集合 (SDK)

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下示例调用 ListCollections 操作。

   Java

   以下示例列出当前区域中的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;

public class ListCollections {

   public static void main(String[] args) throws Exception {


      AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();
```

```
      System.out.println("Listing collections");
      int limit = 10;
      ListCollectionsResult listCollectionsResult = null;
      String paginationToken = null;
      do {
         if (listCollectionsResult != null) {
            paginationToken = listCollectionsResult.getNextToken();
         }
         ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
                 .withMaxResults(limit)
                 .withNextToken(paginationToken);

 listCollectionsResult=amazonRekognition.listCollections(listCollectionsRequest);

         List < String > collectionIds =
 listCollectionsResult.getCollectionIds();
         for (String resultId: collectionIds) {
            System.out.println(resultId);
         }
      } while (listCollectionsResult != null &&
 listCollectionsResult.getNextToken() !=
         null);

   }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.list_collections.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import
 software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
//snippet-end:[rekognition.java2.list_collections.import]

/**
```

```
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ListCollections {

 public static void main(String[] args) {

     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     System.out.println("Listing collections");
     listAllCollections(rekClient);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.list_collections.main]
 public static void listAllCollections(RekognitionClient rekClient) {
     try {
         ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
             .maxResults(10)
             .build();

         ListCollectionsResponse response =
 rekClient.listCollections(listCollectionsRequest);
         List<String> collectionIds = response.collectionIds();
         for (String resultId : collectionIds) {
             System.out.println(resultId);
         }

     } catch (RekognitionException e) {
         System.out.println(e.getMessage());
         System.exit(1);
     }
 }
 // snippet-end:[rekognition.java2.list_collections.main]
```

```
}
```

AWS CLI

此 AWS CLI 命令显示 list-collections CLI 操作的 JSON 输出。将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition list-collections --profile profile-name
```

Python

以下示例列出当前区域中的集合。

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def list_collections():

    max_results=2

    client=boto3.client('rekognition')

    #Display all the collections
    print('Displaying collections...')
    response=client.list_collections(MaxResults=max_results)
    collection_count=0
    done=False

    while done==False:
        collections=response['CollectionIds']

        for collection in collections:
            print (collection)
            collection_count+=1
        if 'NextToken' in response:
            nextToken=response['NextToken']
```

```
 response=client.list_collections(NextToken=nextToken,MaxResults=max_results)

        else:
            done=True

    return collection_count

def main():

    collection_count=list_collections()
    print("collections: " + str(collection_count))
if __name__ == "__main__":
    main()
```

.NET

以下示例列出当前区域中的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class ListCollections
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        ListCollectionsResponse listCollectionsResponse = null;
        String paginationToken = null;
        do
        {
            if (listCollectionsResponse != null)
                paginationToken = listCollectionsResponse.NextToken;
```

```
            ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
            {
                MaxResults = limit,
                NextToken = paginationToken
            };

            listCollectionsResponse =
 rekognitionClient.ListCollections(listCollectionsRequest);

            foreach (String resultId in listCollectionsResponse.CollectionIds)
                Console.WriteLine(resultId);
        } while (listCollectionsResponse != null &&
 listCollectionsResponse.NextToken != null);
    }
}
```

Node.js

将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import { ListCollectionsCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

const listCollection = async () => {
  var max_results = 10
  console.log("Displaying collections:")
```

```
  var response = await rekogClient.send(new ListCollectionsCommand({MaxResults:
 max_results}))
  var collection_count = 0
  var done = false
  while (done == false){
      var collections = response.CollectionIds
      collections.forEach(collection => {
          console.log(collection)
          collection_count += 1
      });
      return collection_count
  }
}

var collect_list = await listCollection()
console.log(collect_list)
```

ListCollections 操作请求

`ListCollections` 的输入是要返回的集合的最大数量。

```
{
    "MaxResults": 2
}
```

如果响应中包含的集合比 `MaxResults` 请求的人脸多，则会返回一个令牌，您可以使用此令牌在后续调用 `ListCollections` 时获得下一组结果。例如：

```
{
    "NextToken": "MGYZLAHX1T5a....",
    "MaxResults": 2
}
```

ListCollections 操作响应

Amazon Rekognition 返回一系列集合（`CollectionIds`）。一个单独的数组 (`FaceModelVersions`) 提供用于分析每个集合中的人脸的人脸模型版本。例如，在下面的 JSON 响应中，集合 `MyCollection` 通过使用 2.0 版本的人脸模型来分析人脸。集合 `AnotherCollection` 使用 3.0 版本的人脸模型。有关更多信息，请参阅 [了解模型版本控制]。

NextToken 是用于在后续调用 ListCollections 时获取下一组结果的令牌。

```
{
    "CollectionIds": [
        "MyCollection",
        "AnotherCollection"
    ],
    "FaceModelVersions": [
        "2.0",
        "3.0"
    ],
    "NextToken": "MGYZLAHX1T5a...."
}
```

描述集合

您可以使用 DescribeCollection 操作获取有关集合的下列信息：

- 在集合中索引的人脸数量。
- 与集合一起使用的模型版本。有关更多信息，请参阅 the section called “了解模型版本控制”。
- 集合的 Amazon 资源名称 (ARN)。
- 集合的创建日期和时间。

描述集合 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DescribeCollection 操作。

   Java

   此示例描述集合。

   将值 collectionId 更改为所需集合的 ID。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DescribeCollectionRequest;
import com.amazonaws.services.rekognition.model.DescribeCollectionResult;


public class DescribeCollection {

   public static void main(String[] args) throws Exception {

      String collectionId = "CollectionID";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      System.out.println("Describing collection: " +
         collectionId );


        DescribeCollectionRequest request = new DescribeCollectionRequest()
                    .withCollectionId(collectionId);

      DescribeCollectionResult describeCollectionResult =
 rekognitionClient.describeCollection(request);
      System.out.println("Collection Arn : " +
         describeCollectionResult.getCollectionARN());
      System.out.println("Face count : " +
         describeCollectionResult.getFaceCount().toString());
      System.out.println("Face model version : " +
         describeCollectionResult.getFaceModelVersion());
      System.out.println("Created : " +
         describeCollectionResult.getCreationTimestamp().toString());

   }

}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.describe_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DescribeCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName>\n\n" +
         "Where:\n" +
         "   collectionName - The name of the Amazon Rekognition collection. \n
\n";

     if (args.length != 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String collectionName = args[0];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
            .build();

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.describe_collection.main]
    public static void describeColl(RekognitionClient rekClient, String
    collectionName) {

        try {
            DescribeCollectionRequest describeCollectionRequest =
    DescribeCollectionRequest.builder()
                .collectionId(collectionName)
                .build();

            DescribeCollectionResponse describeCollectionResponse =
    rekClient.describeCollection(describeCollectionRequest);
            System.out.println("Collection Arn : " +
    describeCollectionResponse.collectionARN());
            System.out.println("Created : " +
    describeCollectionResponse.creationTimestamp().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.describe_collection.main]
}
```

AWS CLI

此 AWS CLI 命令显示 `describe-collection` CLI 操作的 JSON 输出。将 `collection-id` 的值更改为所需集合的 ID。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
aws rekognition describe-collection --collection-id collection-name --profile
 profile-name
```

Python

此示例描述集合。

将值 `collection_id` 更改为所需集合的 ID。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError

def describe_collection(collection_id):

    print('Attempting to describe collection ' + collection_id)

    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

    try:
    response = client.describe_collection(CollectionId=collection_id)
    print("Collection Arn: " + response['CollectionARN'])
    print("Face Count: " + str(response['FaceCount']))
    print("Face Model Version: " + response['FaceModelVersion'])
    print("Timestamp: " + str(response['CreationTimestamp']))

    except ClientError as e:
    if e.response['Error']['Code'] == 'ResourceNotFoundException':
    print('The collection ' + collection_id + ' was not found ')
    else:
    print('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
    print('Done...')

def main():
    collection_id = 'collection-name'
    describe_collection(collection_id)

if __name__ == "__main__":
```

```
    main()
```

.NET

此示例描述集合。

将值 collectionId 更改为所需集合的 ID。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DescribeCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        String collectionId = "CollectionID";
        Console.WriteLine("Describing collection: " + collectionId);

        DescribeCollectionRequest describeCollectionRequest = new
 DescribeCollectionRequest()
        {
            CollectionId = collectionId
        };

        DescribeCollectionResponse describeCollectionResponse =
 rekognitionClient.DescribeCollection(describeCollectionRequest);
        Console.WriteLine("Collection ARN: " +
 describeCollectionResponse.CollectionARN);
        Console.WriteLine("Face count: " +
 describeCollectionResponse.FaceCount);
        Console.WriteLine("Face model version: " +
 describeCollectionResponse.FaceModelVersion);
        Console.WriteLine("Created: " +
 describeCollectionResponse.CreationTimestamp);
```

```
    }
}
```

Node.js

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import { DescribeCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

const describeCollection = async (collectionName) => {
  try {
     console.log(`Attempting to describe collection named - ${collectionName}`)
     var response = await rekogClient.send(new
 DescribeCollectionCommand({CollectionId: collectionName}))
     console.log('Collection Arn:')
     console.log(response.CollectionARN)
     console.log('Face Count:')
     console.log(response.FaceCount)
     console.log('Face Model Version:')
     console.log(response.FaceModelVersion)
     console.log('Timestamp:')
     console.log(response.CreationTimestamp)
     return response; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
```

```
describeCollection(collection_name)
```

DescribeCollection 操作请求

DescribeCollection 的输入是所需集合的 ID，如下面的 JSON 示例所示。

```
{
    "CollectionId": "MyCollection"
}
```

DescribeCollection操作响应

响应包括以下内容：

- 在集合中索引的人脸数量 FaceCount。
- 与集合一起使用的人脸模型版本，FaceModelVersion。有关更多信息，请参阅 the section called “了解模型版本控制”。
- 集合的 Amazon 资源名称 CollectionARN。
- 集合的创建时间和日期 CreationTimestamp。CreationTimestamp 的值是自 Unix 纪元时间到集合创建时间的毫秒数。Unix 纪元时间是 1970 年 1 月 1 日星期四，协调世界时 (UTC) 00:00:00。有关更多信息，请参阅 Unix 时间。

```
{
    "CollectionARN": "arn:aws:rekognition:us-east-1:nnnnnnnnnnnn:collection/
MyCollection",
    "CreationTimestamp": 1.533422155042E9,
    "FaceCount": 200,
    "UserCount" : 20,
    "FaceModelVersion": "1.0"
}
```

删除集合

您可以使用 DeleteCollection 操作删除集合。

有关更多信息，请参阅 管理集合。

删除集合 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DeleteCollection 操作。

   Java

   此示例删除集合。

   将 collectionId 的值更改为您要删除的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteCollectionRequest;
import com.amazonaws.services.rekognition.model.DeleteCollectionResult;


public class DeleteCollection {

   public static void main(String[] args) throws Exception {

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      String collectionId = "MyCollection";

      System.out.println("Deleting collections");

      DeleteCollectionRequest request = new DeleteCollectionRequest()
         .withCollectionId(collectionId);
      DeleteCollectionResult deleteCollectionResult =
 rekognitionClient.deleteCollection(request);
```

```
      System.out.println(collectionId + ": " +
 deleteCollectionResult.getStatusCode()
         .toString());

   }

}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
// snippet-start:[rekognition.java2.delete_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
// snippet-end:[rekognition.java2.delete_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
```

```
            "   <collectionId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection to delete. \n\n";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_collection.main]
    public static void deleteMyCollection(RekognitionClient rekClient,String
 collectionId ) {

        try {
            DeleteCollectionRequest deleteCollectionRequest =
 DeleteCollectionRequest.builder()
                .collectionId(collectionId)
                .build();

            DeleteCollectionResponse deleteCollectionResponse =
 rekClient.deleteCollection(deleteCollectionRequest);
            System.out.println(collectionId + ": " +
 deleteCollectionResponse.statusCode().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.delete_collection.main]
```

```
}
```

AWS CLI

此 AWS CLI 命令显示 delete-collection CLI 操作的 JSON 输出。将 collection-id 的值替换为您要删除的集合的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
aws rekognition delete-collection --collection-id collection-name --profile
 profile-name
```

Python

此示例删除集合。

将 collection_id 的值更改为您要删除的集合。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError

def delete_collection(collection_id):

    print('Attempting to delete collection ' + collection_id)
    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

    status_code = 0

    try:
        response = client.delete_collection(CollectionId=collection_id)
        status_code = response['StatusCode']

    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFoundException':
            print('The collection ' + collection_id + ' was not found ')
        else:
```

```
            print('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
        status_code = e.response['ResponseMetadata']['HTTPStatusCode']
    return (status_code)

def main():

    collection_id = 'collection-name'
    status_code = delete_collection(collection_id)
    print('Status code: ' + str(status_code))

if __name__ == "__main__":
    main()
```

.NET

此示例删除集合。

将 collectionId 的值更改为您要删除的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DeleteCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        String collectionId = "MyCollection";
        Console.WriteLine("Deleting collection: " + collectionId);

        DeleteCollectionRequest deleteCollectionRequest = new
 DeleteCollectionRequest()
        {
            CollectionId = collectionId
```

```
        };

        DeleteCollectionResponse deleteCollectionResponse =
 rekognitionClient.DeleteCollection(deleteCollectionRequest);
        Console.WriteLine(collectionId + ": " +
 deleteCollectionResponse.StatusCode);
    }
}
```

Node.js

将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import { DeleteCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

const deleteCollection = async (collectionName) => {
    try {
       console.log(`Attempting to delete collection named - ${collectionName}`)
       var response = await rekogClient.send(new
 DeleteCollectionCommand({CollectionId: collectionName}))
       var status_code = response.StatusCode
       if (status_code = 200){
           console.log("Collection successfully deleted.")
       }
       return response; // For unit tests.
```

```
    } catch (err) {
      console.log("Error", err.stack);
    }
  };

deleteCollection(collection_name)
```

DeleteCollection 操作请求

DeleteCollection 的输入是要删除的集合的 ID，如下面的 JSON 示例所示。

```
{
    "CollectionId": "MyCollection"
}
```

DeleteCollection 操作响应

DeleteCollection 响应包含 HTTP 状态代码，指示操作成功还是失败。如果成功删除了集合，则返回 200。

```
{"StatusCode":200}
```

将人脸添加到集合

您可以使用 IndexFaces 操作检测图像中的人脸并将人脸添加到集合中。对于检测到的每张人脸，Amazon Rekognition 将提取人脸特征并将特征信息存储到数据库中。此外，该命令将检测到的每个人脸的元数据存储在指定的人脸集合中。Amazon Rekognition 不存储实际的图像字节。

有关提供合适的人脸用于编制索引的信息，请参阅有关面部比较输入图像的建议。

对于每个人脸，IndexFaces 操作保留以下信息：

- 多维人脸特征 – IndexFaces 使用人脸分析来提取有关人脸特征的多维信息，并将该信息存储在人脸集合中。您无法直接访问此信息。不过，Amazon Rekognition 在人脸集合中搜索匹配的人脸时将使用此信息。

- 元数据 — 每张人脸的元数据包括边界框、由 Amazon Rekognition IDs 分配的置信度（边界框包含人脸）（人脸 ID 和图片 ID），以及请求中的外部图片 ID（如果您提供了）。在响应 IndexFaces API 调用时，将为您返回此信息。有关示例，请参阅以下示例响应中的 face 元素。

  该服务将返回此元数据以响应以下 API 调用：

  - ListFaces
  - 搜索人脸操作 - SearchFaces 和 SearchFacesByImage 的响应返回每个匹配人脸的匹配置信度以及已匹配人脸的元数据。

IndexFaces 编入索引的人脸数量取决于与输入集合关联的人脸检测模型的版本。有关更多信息，请参阅 了解模型版本控制。

有关索引的人脸的信息以 FaceRecord 对象数组的形式返回。

您可能希望将索引的人脸与检测到人脸的图像相关联。例如，您可能想要保留图像的客户端索引和图像中的人脸。要将人脸与图像关联，请在 ExternalImageId 请求参数中指定图像 ID。图像 ID 可以是您创建的文件名或其他 ID。

除了 API 在人脸集合中保留的前面的信息之外，API 还返回集合中未保留的人脸详细信息。(请参阅以下示例响应中的 faceDetail 元素)。

> **Note**
>
> DetectFaces 将返回相同的信息，因此您无需对同一张图像调用 DetectFaces 和 IndexFaces。

筛选人脸

该 IndexFaces 操作使您可以过滤从图像中编制索引的面孔。借助 IndexFaces，您可以指定要编制索引的人脸的最大数目，也可以选择仅为检测到的高质量人脸编制索引。

您可以使用 MaxFaces 输入参数指定由 IndexFaces 编制索引的人脸的最大数目。当您想为图像中的最大人脸编制索引而不想对较小人脸（例如，背景中站立的人的脸）编制索引时，这很有用。

默认情况下， IndexFaces 选择用于筛选出人脸的质量条。您可以使用 QualityFilter 输入参数显式设置质量条。值为：

- AUTO – Amazon Rekognition 选择用于筛选出人脸的质量条（默认值）。
- LOW – 除了最低质量人脸之外的所有人脸都将进行索引。
- MEDIUM
- HIGH – 仅对质量最高的人脸进行索引。
- NONE -不会根据质量筛选出任何人脸。

IndexFaces 根据以下条件筛选出人脸：

- 与图像尺寸相比，人脸太小。
- 人脸太模糊。
- 图像太暗。
- 人脸的姿势很极端。
- 人脸没有足够的细节，不适合人脸搜索。

> **Note**
>
> 要使用质量筛选，您需要一个与版本 3 或更高版本的人脸模型关联的集合。要获取与集合关联的人脸模型版本，请调用 DescribeCollection。

有关 IndexFaces 未编制索引的人脸的信息将以 UnindexedFace 对象数组的形式返回。Reasons 数组包含有关未为人脸编制索引的原因的列表。例如，值 EXCEEDS_MAX_FACES 表示未为人脸编制索引，因为检测到的人脸数已达到 MaxFaces 所指定的人脸数。

有关更多信息，请参阅 管理集合中的人脸。

将人脸添加到集合 (SDK)

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 将图像（包含一个或多个人脸）上传到您的 Amazon S3 存储桶。

有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 IndexFaces 操作。

Java

此示例显示添加到集合的人脸的人脸标识符。

将 collectionId 的值更改为您要向其中添加人脸的集合的名称。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。.withMaxFaces(1) 参数将索引的人脸数限制为 1。删除或更改其值以满足您的需求。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.FaceRecord;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.IndexFacesRequest;
import com.amazonaws.services.rekognition.model.IndexFacesResult;
import com.amazonaws.services.rekognition.model.QualityFilter;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.UnindexedFace;
import java.util.List;

public class AddFacesToCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

    public static void main(String[] args) throws Exception {

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        Image image = new Image()
                .withS3Object(new S3Object()
                .withBucket(bucket)
```

```
                .withName(photo));

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
                .withImage(image)
                .withQualityFilter(QualityFilter.AUTO)
                .withMaxFaces(1)
                .withCollectionId(collectionId)
                .withExternalImageId(photo)
                .withDetectionAttributes("DEFAULT");

        IndexFacesResult indexFacesResult =
 rekognitionClient.indexFaces(indexFacesRequest);

        System.out.println("Results for " + photo);
        System.out.println("Faces indexed:");
        List<FaceRecord> faceRecords = indexFacesResult.getFaceRecords();
        for (FaceRecord faceRecord : faceRecords) {
            System.out.println("  Face ID: " +
 faceRecord.getFace().getFaceId());
            System.out.println("  Location:" +
 faceRecord.getFaceDetail().getBoundingBox().toString());
        }

        List<UnindexedFace> unindexedFaces =
 indexFacesResult.getUnindexedFaces();
        System.out.println("Faces not indexed:");
        for (UnindexedFace unindexedFace : unindexedFaces) {
            System.out.println("  Location:" +
 unindexedFace.getFaceDetail().getBoundingBox().toString());
            System.out.println("  Reasons:");
            for (String reason : unindexedFace.getReasons()) {
                System.out.println("   " + reason);
            }
        }
    }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.add_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
```

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.IndexFacesResponse;
import software.amazon.awssdk.services.rekognition.model.IndexFacesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.QualityFilter;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceRecord;
import software.amazon.awssdk.services.rekognition.model.UnindexedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Reason;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.add_faces_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class AddFacesToCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "    <collectionId> <sourceImage>\n\n" +
         "Where:\n" +
         "    collectionName - The name of the collection.\n" +
         "    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length != 2) {
         System.out.println(usage);
         System.exit(1);
     }
```

```
    String collectionId = args[0];
    String sourceImage = args[1];
    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    addToCollection(rekClient, collectionId, sourceImage);
    rekClient.close();
}

// snippet-start:[rekognition.java2.add_faces_collection.main]
public static void addToCollection(RekognitionClient rekClient, String
collectionId, String sourceImage) {

    try {
        InputStream sourceStream = new FileInputStream(sourceImage);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
        Image souImage = Image.builder()
            .bytes(sourceBytes)
            .build();

        IndexFacesRequest facesRequest = IndexFacesRequest.builder()
            .collectionId(collectionId)
            .image(souImage)
            .maxFaces(1)
            .qualityFilter(QualityFilter.AUTO)
            .detectionAttributes(Attribute.DEFAULT)
            .build();

        IndexFacesResponse facesResponse = rekClient.indexFaces(facesRequest);
        System.out.println("Results for the image");
        System.out.println("\n Faces indexed:");
        List<FaceRecord> faceRecords = facesResponse.faceRecords();
        for (FaceRecord faceRecord : faceRecords) {
            System.out.println("  Face ID: " + faceRecord.face().faceId());
            System.out.println("  Location:" +
faceRecord.faceDetail().boundingBox().toString());
        }

        List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
        System.out.println("Faces not indexed:");
        for (UnindexedFace unindexedFace : unindexedFaces) {
```

```
            System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
            System.out.println("  Reasons:");
            for (Reason reason : unindexedFace.reasons()) {
                System.out.println("Reason:  " + reason);
            }
        }

    } catch (RekognitionException | FileNotFoundException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.add_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令显示 index-faces CLI 操作的 JSON 输出。

将 collection-id 的值替换为您希望在其中存储人脸的集合的名称。将Bucket和Name的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像文件。max-faces 参数将索引的人脸数限制为 1。删除或更改其值以满足您的需求。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
aws rekognition index-faces --image '{"S3Object":{"Bucket":"bucket-
name","Name":"file-name"}}' --collection-id "collection-id" \
                                --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
                                --external-image-id "example-image.jpg" --
profile profile-name
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition index-faces --image "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
--external-image-id "example-image.jpg" --profile profile-name
```

Python

此示例显示添加到集合的人脸的人脸标识符。

将 collectionId 的值更改为您要向其中添加人脸的集合的名称。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。MaxFaces 输入参数将索引的人脸数限制为 1。删除或更改其值以满足您的需求。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def add_faces_to_collection(bucket, photo, collection_id):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])

    print('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
        print('  Face ID: ' + faceRecord['Face']['FaceId'])
        print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']
['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])
```

```
def main():
    bucket = 'amzn-s3-demo-bucket'
    collection_id = 'collection-id'
    photo = 'photo-name'

    indexed_faces_count = add_faces_to_collection(bucket, photo, collection_id)
    print("Faces indexed count: " + str(indexed_faces_count))

if __name__ == "__main__":
    main()
```

.NET

此示例显示添加到集合的人脸的人脸标识符。

将 collectionId 的值更改为您要向其中添加人脸的集合的名称。将bucket和photo的值替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class AddFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "amzn-s3-demo-bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        Image image = new Image()
        {
            S3Object = new S3Object()
            {
```

```
                Bucket = bucket,
                Name = photo
            }
        };

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<String>(){ "ALL" }
        };

        IndexFacesResponse indexFacesResponse =
 rekognitionClient.IndexFaces(indexFacesRequest);

        Console.WriteLine(photo + " added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            Console.WriteLine("Face detected: Faceid is " +
               faceRecord.Face.FaceId);
    }
}
```

IndexFaces 操作请求

IndexFaces 的输入是要编入索引的图像和要向其中添加人脸的集合。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "ExternalImageId": "input.jpg",
    "DetectionAttributes": [
        "DEFAULT"
    ],
    "MaxFaces": 1,
    "QualityFilter": "AUTO"
}
```

IndexFaces 操作响应

IndexFaces 返回有关在图像中检测到的人脸的信息。例如，以下 JSON 响应包含在输入图像中检测到的人脸的默认检测属性。此示例还显示未为人脸编制索引，因为已超出 MaxFaces 输入参数的值，Reasons 数组包含 EXCEEDS_MAX_FACES。如果因质量原因而未为人脸编制索引，Reasons 将包含 LOW_SHARPNESS 或 LOW_BRIGHTNESS 等值。有关更多信息，请参阅 UnindexedFace。


```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "ExternalImageId": "input.jpg",
                "FaceId": "b86e2392-9da1-459b-af68-49118dc16f87",
                "ImageId": "09f43d92-02b6-5cea-8fbd-9f187db2050d"
            },
            "FaceDetail": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.5751981735229492,
                        "Y": 0.4010535478591919
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.6511467099189758,
                        "Y": 0.4017036259174347
                    },
                    {
```

```
                    "Type": "nose",
                    "X": 0.6314528584480286,
                    "Y": 0.4710812568664551
                },
                {
                    "Type": "mouthLeft",
                    "X": 0.5879443287849426,
                    "Y": 0.5171778798103333
                },
                {
                    "Type": "mouthRight",
                    "X": 0.6444502472877502,
                    "Y": 0.5164633989334106
                }
            ],
            "Pose": {
                "Pitch": -10.313642501831055,
                "Roll": -1.0316886901855469,
                "Yaw": 18.079818725585938
            },
            "Quality": {
                "Brightness": 71.2919921875,
                "Sharpness": 78.74752044677734
            }
        }
    }
],
"OrientationCorrection": "",
"UnindexedFaces": [
    {
        "FaceDetail": {
            "BoundingBox": {
                "Height": 0.1329464465379715,
                "Left": 0.5611110925674438,
                "Top": 0.6832437515258789,
                "Width": 0.08777777850627899
            },
            "Confidence": 92.37225341796875,
            "Landmarks": [
                {
                    "Type": "eyeLeft",
                    "X": 0.5796897411346436,
                    "Y": 0.7452847957611084
                },
```

```
                    {
                        "Type": "eyeRight",
                        "X": 0.6078574657440186,
                        "Y": 0.742687463760376
                    },
                    {
                        "Type": "nose",
                        "X": 0.597953200340271,
                        "Y": 0.7620673179626465
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.5884202122688293,
                        "Y": 0.7920381426811218
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.60627681016922,
                        "Y": 0.7919750809669495
                    }
                ],
                "Pose": {
                    "Pitch": 15.658954620361328,
                    "Roll": -4.583454608917236,
                    "Yaw": 10.558992385864258
                },
                "Quality": {
                    "Brightness": 42.54612350463867,
                    "Sharpness": 86.93206024169922
                }
            },
            "Reasons": [
                "EXCEEDS_MAX_FACES"
            ]
        }
    ]
}
```


要获取所有面部信息，对于 DetectionAttributes 请求参数，请指定“ALL”。例如，在以下示例响应中，记住 faceDetail 元素中的其他信息，这些信息不会保留在服务器上：

- 25 个人脸标记（相较于上一个示例中的仅 5 个人脸标记）
- 十个人脸属性（眼镜、胡须、遮挡、视线方向等）

- 情绪（请参阅 emotion 元素）

face 元素提供了服务器上保留的元数据。

FaceModelVersion 是与集合关联的人脸模型的版本。有关更多信息，请参阅 了解模型版本控制。

OrientationCorrection 是估计的图像方向。如果您使用的是高于版本 3 的人脸检测模型版本，则不会返回方向校正信息。有关更多信息，请参阅 获取图像方向和边界框坐标。

以下示例响应显示了指定 ["ALL"] 时返回的 JSON：


```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333333253860474,
                    "Left": 0.17185185849666595,
                    "Top": 0.7366666793823242,
                    "Width": 0.11061728745698929
                },
                "Confidence": 99.99999237060547,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "FaceDetail": {
                "AgeRange": {
                    "High": 25,
                    "Low": 15
                },
                "Beard": {
                    "Confidence": 99.98077392578125,
                    "Value": false
                },
                "BoundingBox": {
                    "Height": 0.06333333253860474,
                    "Left": 0.17185185849666595,
                    "Top": 0.7366666793823242,
                    "Width": 0.11061728745698929
                },
                "Confidence": 99.99999237060547,
```

```
                "Emotions": [
                    {
                        "Confidence": 95.40877532958984,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 6.6088080406188965,
                        "Type": "CALM"
                    },
                    {
                        "Confidence": 0.7385611534118652,
                        "Type": "SAD"
                    }
                ],
                 "EyeDirection": {
                    "yaw": 16.299732,
                    "pitch": -6.407457,
                    "confidence": 99.968704
                              }
                "Eyeglasses": {
                    "Confidence": 99.96795654296875,
                    "Value": false
                },
                "EyesOpen": {
                    "Confidence": 64.0671157836914,
                    "Value": true
                },
                "Gender": {
                    "Confidence": 100,
                    "Value": "Female"
                },
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.21361233294010162,
                        "Y": 0.757106363773346
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.2518567442893982,
                        "Y": 0.7599404454231262
                    },
                    {
                        "Type": "nose",
```

```
                    "X": 0.2262365221977234,
                    "Y": 0.7711842060089111
                },
                {
                    "Type": "mouthLeft",
                    "X": 0.2050037682056427,
                    "Y": 0.7801263332366943
                },
                {
                    "Type": "mouthRight",
                    "X": 0.2430567592382431,
                    "Y": 0.7836716771125793
                },
                {
                    "Type": "leftPupil",
                    "X": 0.2161938101053238,
                    "Y": 0.756662905216217
                },
                {
                    "Type": "rightPupil",
                    "X": 0.2523181438446045,
                    "Y": 0.7603650689125061
                },
                {
                    "Type": "leftEyeBrowLeft",
                    "X": 0.20066319406032562,
                    "Y": 0.7501518130302429
                },
                {
                    "Type": "leftEyeBrowUp",
                    "X": 0.2130996286869049,
                    "Y": 0.7480520606040955
                },
                {
                    "Type": "leftEyeBrowRight",
                    "X": 0.22584207355976105,
                    "Y": 0.7504606246948242
                },
                {
                    "Type": "rightEyeBrowLeft",
                    "X": 0.24509544670581818,
                    "Y": 0.7526801824569702
                },
                {
```

```
                    "Type": "rightEyeBrowUp",
                    "X": 0.2582615911960602,
                    "Y": 0.7516844868659973
                },
                {
                    "Type": "rightEyeBrowRight",
                    "X": 0.26881539821624756,
                    "Y": 0.7554477453231812
                },
                {
                    "Type": "leftEyeLeft",
                    "X": 0.20624476671218872,
                    "Y": 0.7568746209144592
                },
                {
                    "Type": "leftEyeRight",
                    "X": 0.22105035185813904,
                    "Y": 0.7582521438598633
                },
                {
                    "Type": "leftEyeUp",
                    "X": 0.21401576697826385,
                    "Y": 0.7553104162216187
                },
                {
                    "Type": "leftEyeDown",
                    "X": 0.21317370235919952,
                    "Y": 0.7584449648857117
                },
                {
                    "Type": "rightEyeLeft",
                    "X": 0.24393919110298157,
                    "Y": 0.7600628137588501
                },
                {
                    "Type": "rightEyeRight",
                    "X": 0.2598416209220886,
                    "Y": 0.7605880498886108
                },
                {
                    "Type": "rightEyeUp",
                    "X": 0.2519053518772125,
                    "Y": 0.7582084536552429
                },
```

```
                    {
                        "Type": "rightEyeDown",
                        "X": 0.25177454948425293,
                        "Y": 0.7612871527671814
                    },
                    {
                        "Type": "noseLeft",
                        "X": 0.2185886949300766,
                        "Y": 0.774715781211853
                    },
                    {
                        "Type": "noseRight",
                        "X": 0.23328955471515656,
                        "Y": 0.7759330868721008
                    },
                    {
                        "Type": "mouthUp",
                        "X": 0.22446128726005554,
                        "Y": 0.7805567383766174
                    },
                    {
                        "Type": "mouthDown",
                        "X": 0.22087252140045166,
                        "Y": 0.7891407608985901
                    }
                ],
                "MouthOpen": {
                    "Confidence": 95.87068939208984,
                    "Value": false
                },
                "Mustache": {
                    "Confidence": 99.9828109741211,
                    "Value": false
                },
                "Pose": {
                    "Pitch": -0.9409101605415344,
                    "Roll": 7.233824253082275,
                    "Yaw": -2.3602254390716553
                },
                "Quality": {
                    "Brightness": 32.01998519897461,
                    "Sharpness": 93.67259216308594
                },
                "Smile": {
```

```
                    "Confidence": 86.7142105102539,
                    "Value": true
                },
                "Sunglasses": {
                    "Confidence": 97.38925170898438,
                    "Value": false
                }
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
    "UnindexedFaces": []
}
```

列出集合中的人脸和相关用户

您可以使用该ListFaces操作在集合中列出面孔及其关联用户。

有关更多信息，请参阅 管理集合中的人脸。

列出集合中的人脸 (SDK)

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下示例调用 ListFaces 操作。

   Java

   此示例显示了集合中的人脸列表。

   将 collectionId 的值更改为所需的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.ListFacesRequest;
import com.amazonaws.services.rekognition.model.ListFacesResult;
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;



public class ListFacesInCollection {
    public static final String collectionId = "MyCollection";

   public static void main(String[] args) throws Exception {

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

      ListFacesResult listFacesResult = null;
      System.out.println("Faces in collection " + collectionId);

      String paginationToken = null;
      do {
         if (listFacesResult != null) {
            paginationToken = listFacesResult.getNextToken();
         }

         ListFacesRequest listFacesRequest = new ListFacesRequest()
                 .withCollectionId(collectionId)
                 .withMaxResults(1)
                 .withNextToken(paginationToken);

         listFacesResult =  rekognitionClient.listFaces(listFacesRequest);
         List < Face > faces = listFacesResult.getFaces();
         for (Face face: faces) {
            System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
               .writeValueAsString(face));
         }
      } while (listFacesResult != null && listFacesResult.getNextToken() !=
         null);
   }
```

```
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
// snippet-start:[rekognition.java2.list_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.list_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListFacesInCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId>\n\n" +
            "Where:\n" +
            "   collectionId - The name of the collection. \n\n";

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.list_faces_collection.main]
    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId ) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                .collectionId(collectionId)
                .maxResults(10)
                .build();

            ListFacesResponse facesResponse = rekClient.listFaces(facesRequest);
            List<Face> faces = facesResponse.faces();
            for (Face face: faces) {
                System.out.println("Confidence level there is a face:
 "+face.confidence());
                System.out.println("The face Id value is "+face.faceId());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
         }
      }
    // snippet-end:[rekognition.java2.list_faces_collection.main]
  }
```

AWS CLI

此 AWS CLI 命令显示 `list-faces` CLI 操作的 JSON 输出。将 `collection-id` 的值替换为您要列出的集合的名称。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
aws rekognition list-faces --collection-id "collection-id"  --profile profile-
name
```

Python

此示例显示了集合中的人脸列表。

将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def list_faces_in_collection(collection_id):
    maxResults = 2
    faces_count = 0
    tokens = True

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    response = client.list_faces(CollectionId=collection_id,
                                 MaxResults=maxResults)

    print('Faces in collection ' + collection_id)

    while tokens:

        faces = response['Faces']

        for face in faces:
            print(face)
            faces_count += 1
        if 'NextToken' in response:
            nextToken = response['NextToken']
            response = client.list_faces(CollectionId=collection_id,
                                         NextToken=nextToken,
 MaxResults=maxResults)
        else:
            tokens = False
    return faces_count
```

```
def main():
    collection_id = 'collection-id'
    faces_count = list_faces_in_collection(collection_id)
    print("faces count: " + str(faces_count))

if __name__ == "__main__":
    main()
```

.NET

此示例显示了集合中的人脸列表。

将 collectionId 的值更改为所需的集合。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class ListFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        ListFacesResponse listFacesResponse = null;
        Console.WriteLine("Faces in collection " + collectionId);

        String paginationToken = null;
        do
        {
            if (listFacesResponse != null)
                paginationToken = listFacesResponse.NextToken;

            ListFacesRequest listFacesRequest = new ListFacesRequest()
            {
```

```
                CollectionId = collectionId,
                MaxResults = 1,
                NextToken = paginationToken
            };

            listFacesResponse = rekognitionClient.ListFaces(listFacesRequest);
            foreach(Face face in listFacesResponse.Faces)
                Console.WriteLine(face.FaceId);
        } while (listFacesResponse != null && !
String.IsNullOrEmpty(listFacesResponse.NextToken));
    }
}
```

ListFaces 操作请求

的输入`ListFaces`是您要为其列出面孔的集合的 ID。 `MaxResults`是要返回的最大面孔数。
ListFaces 还会接收 IDs 用于筛选结果的人脸列表，以及用于仅列出与给定用户关联的面孔的用户 ID。

```
{
    "CollectionId": "MyCollection",
    "MaxResults": 1
}
```

如果响应中包含的人脸比 `MaxResults` 请求的人脸多，则会返回一个令牌，您可以使用此令牌在后续调用 `ListFaces` 时获得下一组结果。例如：

```
{
    "CollectionId": "MyCollection",
    "NextToken": "sm+5ythT3aeEVIR4WA....",
    "MaxResults": 1
}
```

ListFaces 操作响应

`ListFaces` 中的响应是与指定集合中存储的人脸元数据有关的信息。

- FaceModelVersion— 与该系列关联的脸部模型的版本。有关更多信息，请参阅 了解模型版本控制。
- Faces – 有关集合中人脸的信息。这包括有关 BoundingBox、置信度、图像标识符和人脸 ID 的信息。有关更多信息，请参阅人脸。
- NextToken— 用于获取下一组结果的标记。

```
{
    "FaceModelVersion": "6.0",
    "Faces": [
        {
            "Confidence": 99.76940155029297,
            "IndexFacesModelVersion": "6.0",
            "UserId": "demoUser2",
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65",
            "BoundingBox": {
                "Width": 0.03177810087800026,
                "Top": 0.36568498611450195,
                "Left": 0.3453829884529114,
                "Height": 0.056759100407361984
            },
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        },
        {
            "BoundingBox": {
                "Width": 0.03254450112581253,
                "Top": 0.6080359816551208,
                "Left": 0.5160620212554932,
                "Height": 0.06347999721765518
            },
            "IndexFacesModelVersion": "6.0",
            "FaceId": "851cb847-dccc-4fea-9309-9f4805967855",
            "Confidence": 99.94369506835938,
            "ImageId": "a8aed589-ceec-35f7-9c04-82e0b546b024"
        },
        {
            "BoundingBox": {
                "Width": 0.03094629943370819,
                "Top": 0.4218429923057556,
                "Left": 0.6513839960098267,
                "Height": 0.05266290158033371
            },
            "IndexFacesModelVersion": "6.0",
            "FaceId": "c0eb3b65-24a0-41e1-b23a-1908b1aaeac1",
            "Confidence": 99.82969665527344,
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65"
        }
    ]
}
```

从集合中删除人脸

您可以使用 DeleteFaces 操作从集合中删除人脸。有关更多信息，请参阅 管理集合中的人脸。

从集合中删除人脸

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下示例调用 DeleteFaces 操作。

   Java

   此示例从集合中删除单个人脸。

   将 collectionId 的值更改为包含您要删除的人脸的集合。将 faces 的值更改为您要删除的人脸的 ID。要删除多张脸，请将该人脸 IDs 添加到faces数组中。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteFacesRequest;
import com.amazonaws.services.rekognition.model.DeleteFacesResult;

import java.util.List;


public class DeleteFacesFromCollection {
   public static final String collectionId = "MyCollection";
   public static final String faces[] = {"xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"};

   public static void main(String[] args) throws Exception {
```

```
      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
              .withCollectionId(collectionId)
              .withFaceIds(faces);

      DeleteFacesResult
 deleteFacesResult=rekognitionClient.deleteFaces(deleteFacesRequest);


      List < String > faceRecords = deleteFacesResult.getDeletedFaces();
      System.out.println(Integer.toString(faceRecords.size()) + " face(s)
 deleted:");
      for (String face: faceRecords) {
         System.out.println("FaceID: " + face);
      }
   }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
// snippet-end:[rekognition.java2.delete_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <faceId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection from which faces are
 deleted. \n\n" +
            "   faceId - The id of the face to delete. \n\n";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_faces_collection.main]
    public static void deleteFacesCollection(RekognitionClient rekClient,
                                             String collectionId,
                                             String faceId) {

        try {
            DeleteFacesRequest deleteFacesRequest = DeleteFacesRequest.builder()
                .collectionId(collectionId)
                .faceIds(faceId)
                .build();
```

```
            rekClient.deleteFaces(deleteFacesRequest);
            System.out.println("The face was deleted from the collection.");

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.delete_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令显示 delete-faces CLI 操作的 JSON 输出。将 collection-id 的值替换为您要删除的人脸所在集合的名称。将的face-ids值替换为要删除 IDs 的人脸数组。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
aws rekognition delete-faces --collection-id "collection-id" --face-ids "faceid"
 --profile profile-name
```

Python

此示例从集合中删除单个人脸。

将 collectionId 的值更改为包含您要删除的人脸的集合。将 faces 的值更改为您要删除的人脸的 ID。要删除多张脸，请将该人脸 IDs 添加到faces数组中。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def delete_faces_from_collection(collection_id, faces):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    response = client.delete_faces(CollectionId=collection_id,
                                   FaceIds=faces)
```

```
    print(str(len(response['DeletedFaces'])) + ' faces deleted:')
    for faceId in response['DeletedFaces']:
        print(faceId)
    return len(response['DeletedFaces'])


def main():
    collection_id = 'collection-id'
    faces = []
    faces.append("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")

    faces_count = delete_faces_from_collection(collection_id, faces)
    print("deleted faces count: " + str(faces_count))

if __name__ == "__main__":
    main()
```

.NET

此示例从集合中删除单个人脸。

将 collectionId 的值更改为包含您要删除的人脸的集合。将 faces 的值更改为您要删除的人脸的 ID。要删除多张面孔，请将该人脸 IDs 添加到faces列表中。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DeleteFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        List<String> faces = new List<String>() { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" };

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();
```

```
        DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
        {
            CollectionId = collectionId,
            FaceIds = faces
        };

        DeleteFacesResponse deleteFacesResponse =
 rekognitionClient.DeleteFaces(deleteFacesRequest);
        foreach (String face in deleteFacesResponse.DeletedFaces)
            Console.WriteLine("FaceID: " + face);
    }
}
```

DeleteFaces 操作请求

的输入DeleteFaces是包含面孔的集合的 ID，以及要删除的人脸 IDs 的数组。

```
{
    "CollectionId": "MyCollection",
    "FaceIds": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

DeleteFaces 操作响应

DeleteFaces响应包含已删除人脸 IDs 的面孔数组。

```
{
    "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

如果输入中 IDs 提供的面孔当前与用户关联，则将以正当理由将其作为其中的 UnsuccessfulFaceDeletions 一部分返回。

```
{
   "DeletedFaces": [
```

```
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ],
     "UnsuccessfulFaceDeletions" : [
        {
           "FaceId" : "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
           "UserId" : "demoUser1",
           "Reason" : ["ASSOCIATED_TO_AN_EXISTING_USER"]
        }
     ]
}
```

创建用户

您可以使用该CreateUser操作使用您提供的唯一用户 ID 在集合中创建新用户。然后，您可以将多张人脸与新创建的用户相关联。

创建用户 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新 IAM 用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 CreateUser 操作。

   Java

   此 Java 代码示例创建了一名用户。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateUserRequest;
import com.amazonaws.services.rekognition.model.CreateUserResult;


public class CreateUser {

    public static void main(String[] args) throws Exception {
```

```
        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        //Replace collectionId and userId with the name of the user that you
 want to create in that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Creating new user: " +
                userId);

        CreateUserRequest request = new CreateUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.createUser(request);
    }

}
```

AWS CLI

此 AWS CLI 命令使用 `create-user` CLI 操作创建用户。

```
aws rekognition create-user --user-id user-id --collection-id collection-name --
region region-name
--client-request-token request-token
```

Python

此 Python 代码示例创建了一名用户。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')
```

```
def create_user(collection_id, user_id):
    """
    Creates a new User within a collection specified by CollectionId.
    Takes UserId as a parameter, which is a user provided ID which
    should be unique within the collection.

    :param collection_id: The ID of the collection where the indexed faces will
 be stored at.
    :param user_id: ID for the UserID to be created. This ID needs to be unique
 within the collection.

    :return: The indexFaces response
    """
    try:
        logger.info(f'Creating user: {collection_id}, {user_id}')
        client.create_user(
            CollectionId=collection_id,
            UserId=user_id
        )
    except ClientError:
        logger.exception(f'Failed to create user with given user id:
 {user_id}')
        raise

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    create_user(collection_id, user_id)

if __name__ == "__main__":
    main()
```

Go

此 Go 代码示例使用 AWS Go SDK V2 并创建了一个用户。

```
package main

import (
 "context"
 "fmt"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rekognition"
)

func main() {
 // Load the AWS SDK configuration
 cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-
west-2"))
 if err != nil {
  log.Fatalf("Failed to load configuration: %v", err)
 }

 // Create a Rekognition client
 client := rekognition.NewFromConfig(cfg)

 // Set up the input parameters
 input := &rekognition.CreateUserInput{
  CollectionId: aws.String("my-new-collection"), // Replace with your collection
 ID
  UserId:       aws.String("user12345678910"),   // Replace with desired user ID
 }

 // Call the CreateUser operation
 result, err := client.CreateUser(context.TODO(), input)
 if err != nil {
  log.Fatalf("Failed to create user: %v", err)
 }

 // Print out the results
 fmt.Printf("User created successfully:\n")

}
```

删除用户

您可以使用该DeleteUser操作根据提供的用户 ID 从集合中删除用户。请注意，在删除指定的用户 ID 之前，与该用户 ID 关联的所有人脸都将与用户 ID 断开关联。

删除用户 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新 IAM 用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DeleteUser 操作。

   Java

   此 Java 代码示例删除了一名用户。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteUserRequest;
import com.amazonaws.services.rekognition.model.DeleteUserResult;


public class DeleteUser {

    public static void main(String[] args) throws Exception {


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        //Replace collectionId and userId with the name of the user that you
 want to delete from that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Deleting existing user: " +
                userId);

        DeleteUserRequest request = new DeleteUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.deleteUser(request);
    }
```

```
}
```

AWS CLI

此 AWS CLI 命令使用 `create-user` CLI 操作删除用户。

```
aws rekognition delete-user --collection-id MyCollection
--user-id user-id --collection-id collection-name --region region-name
```

Python

此 Python 代码示例删除了一名用户。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def delete_user(collection_id, user_id):
        """
        Delete the user from the given collection

        :param collection_id: The ID of the collection where user is stored.
        :param user_id: The ID of the user in the collection to delete.
        """
        logger.info(f'Deleting user: {collection_id}, {user_id}')
        try:
            client.delete_user(
                CollectionId=collection_id,
                UserId=user_id
            )
        except ClientError:
            logger.exception(f'Failed to delete user with given user id:
 {user_id}')
            raise
```

```
def main():
    collection_id = "collection-id"
    user_id = "user-id"
    delete_user(collection_id, user_id)

if __name__ == "__main__":
    main()
```

将人脸与用户关联

您可以使用该AssociateFaces操作将多个单独的面孔与单个用户相关联。要将人脸与用户关联，必须先创建一个集合和一名用户。请注意，人脸向量必须位于用户向量所在的同一个集合中。

关联人脸 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 AssociateFaces 操作。

   Java

   此 Java 代码示例将人脸与用户关联起来。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AssociateFacesRequest;
import com.amazonaws.services.rekognition.model.AssociateFacesResult;


public class AssociateFaces {

    public static void main(String[] args) throws Exception {
```

```
        AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

        /* Replace the below configurations to allow you successfully run the
example

           @collectionId: The collection where user and faces are stored
           @userId: The user which faces will get associated to
           @faceIds: The list of face IDs that will get associated to the given
user
           @userMatchThreshold: Minimum User match confidence required for the
face to
                                be associated with a User that has at least one
faceID already associated
         */

        String collectionId = "MyCollection";
        String userId = "demoUser";
        String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        List<String> faceIds = Arrays.asList(faceid1,faceid2);

        float userMatchThreshold = 0f;
        System.out.println("Associating faces to the existing user: " +
                userId);

        AssociateFacesRequest request = new AssociateFacesRequest()
                .withCollectionId(collectionId)
                .withUserId(userId)
                .withFaceIds(faceIds)
                .withUserMatchThreshold(userMatchThreshold);

        AssociateFacesResult result = rekognitionClient.associateFaces(request);

        System.out.println("Successful face associations: " +
result.getAssociatedFaces().size());
        System.out.println("Unsuccessful face associations: " +
result.getUnsuccessfulFaceAssociations().size());
    }

}
```

AWS CLI

此 AWS CLI 命令使用 associate-faces CLI 操作将人脸与用户关联起来。

```
aws rekognition associate-faces --user-id user-id --face-ids face-id-1 face-id-2
--collection-id collection-name
--region region-name
```

Python

此 Python 代码示例将人脸与用户关联起来。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def associate_faces(collection_id, user_id, face_ids):
    """
    Associate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to associate faces to
    :param face_ids: The list of face IDs to be associated to the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Associating faces to user: {user_id}, {face_ids}')
    try:
        response = client.associate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- associated {len(response["AssociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to associate faces to the given user")
        raise
```

```
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    associate_faces(collection_id, user_id, face_ids)

if __name__ == "__main__":
    main()
```

AssociateFaces 操作响应

AssociateFaces 的响应包括 UserStatus（即解除关联请求的状态）以及要关联的 FaceIds 的列表。还会返回一个 UnsuccessfulFaceAssociations 列表。向 AssociateFaces 提交请求后，操作可能需要一分钟左右的时间才能完成。

因此，将返回 UserStatus ，其值可能如下所示：

- CREATED - 表示“用户”已成功创建，并且当前没有人脸与之关联。在进行任何成功的 “” 呼叫之前，“用户AssociateFaces” 将处于此状态。
- UPDATING - 表示正在更新“用户”以反映新关联/取消关联的面孔，并且将在几秒钟后变为活动状态。搜索结果可能包含处于这种状态的“用户”，客户可以选择在返回的结果中忽略他们。
- ACTIVE - 表示“用户”已更新以反映所有关联/已取消关联的面孔，并且处于可搜索状态。

```
{
    "UnsuccessfulFaceAssociations": [
        {
            "Reasons": [
                "LOW_MATCH_CONFIDENCE"
            ],
            "FaceId": "f5817d37-94f6-0000-bfee-1a2b3c4d5e6f",
            "Confidence": 0.9375374913215637
        },
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
```

```
            "FaceId": "851cb847-dccc-1111-bfee-1a2b3c4d5e6f",
            "UserId": "demoUser2"
        }
    ],
    "UserStatus": "UPDATING",
    "AssociatedFaces": [
        {
            "FaceId": "35ebbb41-7f67-2222-bfee-1a2b3c4d5e6f"
        }
    ]
}
```

取消用户人脸关联

您可以使用该DisassociateFaces操作来移除用户 ID 和面容 ID 之间的关联。

取消关联人脸 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DisassociateFaces 操作。

   Java

   此 Java 示例删除了 FaceID 和用户 ID 与 DisassociateFaces 操作之间的关联。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DisassociateFacesRequest;
import com.amazonaws.services.rekognition.model.DisassociateFacesResult;


public class DisassociateFaces {
```

```
   public static void main(String[] args) throws Exception {


       AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

       /* Replace the below configurations to allow you successfully run the
example

          @collectionId: The collection where user and faces are stored
          @userId: The user which faces will get disassociated from
          @faceIds: The list of face IDs that will get disassociated from the
given user
        */

       String collectionId = "MyCollection";
       String userId = "demoUser";
       String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       List<String> faceIds = Arrays.asList(faceid1,faceid2);

       System.out.println("Disassociating faces from existing user: " +
               userId);

       DisassociateFacesRequest request = new DisassociateFacesRequest()
               .withCollectionId(collectionId)
               .withUserId(userId)
               .withFaceIds(faceIds)

       DisassociateFacesResult result =
rekognitionClient.disassociateFaces(request);

       System.out.println("Successful face disassociations: " +
result.getDisassociatedFaces().size());
       System.out.println("Unsuccessful face disassociations: " +
result.getUnsuccessfulFaceDisassociations().size());
   }

}
```

AWS CLI

此 AWS CLI 命令会移除 FaceID 和用户 ID 与DisassociateFaces操作之间的关联。

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
--user-id user-id --collection-id collection-name --region region-name
```

Python

以下示例删除了 FaceID 和用户 ID 与 DisassociateFaces 操作之间的关联。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given
 user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
```

```
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

if __name__ == "__main__":
    main()
```

DisassociateFaces 操作响应

DisassociateFaces 的响应包括 UserStatus（即解除关联请求的状态）以及要取消关联的 FaceIds 的列表。还会返回一个 UnsuccessfulFaceDisassociations 列表。向提交请求后 DisassociateFaces，操作可能需要一分钟左右的时间才能完成。因此，将返回 UserStatus ，其值可能如下所示：

- CREATED - 表示“用户”已成功创建，并且当前没有人脸与之关联。在进行任何成功的 “” 呼叫之前，“用户AssociateFaces” 将处于此状态。
- UPDATING - 表示正在更新“用户”以反映新关联/取消关联的面孔，并且将在几秒钟后变为活动状态。搜索结果可能包含处于这种状态的“用户”，客户可以选择在返回的结果中忽略他们。
- ACTIVE - 表示“用户”已更新以反映所有关联/已取消关联的面孔，并且处于可搜索状态。

```
{
    "UserStatus": "UPDATING",
    "DisassociatedFaces": [
        {
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        }
    ],
    "UnsuccessfulFaceDisassociations": [
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
            "FaceId": "f5817d37-94f6-4335-bfee-6cf79a3d806e",
```

```
            "UserId": "demoUser1"
        }
    ]
}
```

列出集合中的用户

您可以使用ListUsers操作来列出 UserIds 和 UserStatus. 要查看与用户 ID 关联IDs 的面孔，请使用操作。ListFaces

列出用户 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 ListUsers 操作。

   Java

   此 Java 示例使用 ListUsers 操作列出了集合中的用户。

```
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListUsersRequest;
import com.amazonaws.services.rekognition.model.ListUsersResult;
import com.amazonaws.services.rekognition.model.User;

public class ListUsers {

    public static void main(String[] args) throws Exception {


        AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


        System.out.println("Listing users");
```

```
        int limit = 10;
        ListUsersResult listUsersResult = null;
        String paginationToken = null;
        do {
            if (listUsersResult != null) {
                paginationToken = listUsersResult.getNextToken();
            }
            ListUsersRequest request = new ListUsersRequest()
                    .withCollectionId(collectionId)
                    .withMaxResults(limit)
                    .withNextToken(paginationToken);
            listUsersResult = amazonRekognition.listUsers(request);

            List<User> users = listUsersResult.getUsers();
            for (User currentUser: users) {
                System.out.println(currentUser.getUserId() + " : " +
 currentUser.getUserStatus());
            }
        } while (listUsersResult.getNextToken() != null);
    }
}
```

AWS CLI

此 AWS CLI 命令列出集合中包含该ListUsers操作的用户。

```
aws rekognition list-users --collection-id collection-id --max-results number-
of-max-results
```

Python

以下示例使用 ListUsers 操作列出了集合中的用户。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
import logging
from pprint import pprint

logger = logging.getLogger(__name__)
```

```
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')


def list_users(collection_id):
    """
    List all users from the given collection

    :param collection_id: The ID of the collection where user is stored.

    :return: response that contains list of Users found within given collection
    """
    logger.info(f'Listing the users in collection: {collection_id}')
    try:
        response = client.list_users(
            CollectionId=collection_id
        )
        pprint(response["Users"])
    except ClientError:
        logger.exception(f'Failed to list all user from given collection:
 {collection_id}')
        raise
    else:
        return response

def main():
    collection_id = "collection-id"
    list_users(collection_id)

if __name__ == "__main__":
    main()
```

ListUsers 操作响应

对请求的响应， ListUsers 包括集合`Users`中的列表以及用户的`UsedId`和`UserStatus`。

```
{
    "NextToken": "B1asJT3bAb/ttuGgPFV8BZoBZyGQzlUHXbuTNLh48a6enU7kXKw43hpOwizW7LOk/
Gk7Em09lznoq6+FcDCcSq2olrn7A98BLkt5keu+ZRVRUTyrXtT6J7Hmp
+ieQ2an6Zu0qzPfcdPeaJ9eAxG2d0WNrzJgi5hvmjoiSTTfKX3MQz1sduWQkvAAs4hZfhZoKFahFlqWofshCXa/
FHAAY3PL1PjxXbkNeSSMq8V7i1MlKCdrPVykCv9MokpPt7jtNvKPEZGUhxgBTFMxNWLEcFnzAiCWDg91dFy/
```

```
LalshPjXA9UVc5Gx9vIJNQ/
eO3cQRghAkCT3FOAiXsLAnA015ODTomZpWWVpqB21wKpI3LYmfAVFrDPGzpbTVlRmLsJm41bkmnBBBw9+DHz1Jn7zW
+qc5Fs3yaHu0f51Xg==",
    "Users": [
        {
            "UserId": "demoUser4",
            "UserStatus": "CREATED"
        },
        {
            "UserId": "demoUser2",
            "UserStatus": "CREATED"
        }
    ]
}
```

使用人脸 ID 搜索人脸

您可以使用该SearchFaces操作在集合中搜索与所提供图像中最大面孔匹配的用户。

当检测到人脸并将人脸添加到集合时，IndexFaces 操作响应中将返回人脸 ID。有关更多信息，请参阅管理集合中的人脸。

使用人脸 ID 搜索集合中的相应人脸 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 SearchFaces 操作。

   Java

   此示例显示与通过 ID 标识的人脸匹配的人脸的相关信息。

   将 collectionID 的值更改为包含所需人脸的集合。将 faceId 的值更改为您要查找的人脸的标识符。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.SearchFacesRequest;
import com.amazonaws.services.rekognition.model.SearchFacesResult;
import java.util.List;


  public class SearchFaceMatchingIdCollection {
      public static final String collectionId = "MyCollection";
      public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx";

    public static void main(String[] args) throws Exception {

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        ObjectMapper objectMapper = new ObjectMapper();
      // Search collection for faces matching the face id.

      SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
              .withCollectionId(collectionId)
              .withFaceId(faceId)
              .withFaceMatchThreshold(70F)
              .withMaxFaces(2);

       SearchFacesResult searchFacesByIdResult =
               rekognitionClient.searchFaces(searchFacesRequest);

       System.out.println("Face matching faceId " + faceId);
      List < FaceMatch > faceImageMatches =
 searchFacesByIdResult.getFaceMatches();
      for (FaceMatch face: faceImageMatches) {
         System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
                 .writeValueAsString(face));
```

```
            System.out.println();
        }
    }

}
```

运行示例代码。将显示有关匹配的人脸的信息。

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
// snippet-start:[rekognition.java2.match_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.match_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingIdCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection.  \n" +
```

```
            "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.match_faces_collection.main]
    public static void searchFacebyId(RekognitionClient rekClient,String
 collectionId, String faceId) {

        try {
            SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
                .collectionId(collectionId)
                .faceId(faceId)
                .faceMatchThreshold(70F)
                .maxFaces(2)
                .build();

            SearchFacesResponse imageResponse =
 rekClient.searchFaces(searchFacesRequest) ;
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face: faceImageMatches) {
                System.out.println("The similarity level is
 "+face.similarity());
                System.out.println();
            }
```

```
        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.match_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令显示 search-faces CLI 操作的 JSON 输出。将 face-id 的值替换为您要搜索的人脸标识符，并将 collection-id 的值替换为您要在其中执行搜索的集合。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
aws rekognition search-faces --face-id face-id --collection-id "collection-id"
 --profile profile-name
```

Python

此示例显示与通过 ID 标识的人脸匹配的人脸的相关信息。

将 collectionID 的值更改为包含所需人脸的集合。将 faceId 的值更改为您要查找的人脸的标识符。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def search_face_in_collection(face_id, collection_id):
    threshold = 90
    max_faces = 2

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.search_faces(CollectionId=collection_id,
                                   FaceId=face_id,
                                   FaceMatchThreshold=threshold,
                                   MaxFaces=max_faces)
```

```
    face_matches = response['FaceMatches']
    print('Matching faces')
    for match in face_matches:
        print('FaceId:' + match['Face']['FaceId'])
        print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")

    return len(face_matches)

def main():
    face_id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
    collection_id = 'collection-id'

    faces = []
    faces.append(face_id)

    faces_count = search_face_in_collection(face_id, collection_id)
    print("faces found: " + str(faces_count))

if __name__ == "__main__":
    main()
```

.NET

此示例显示与通过 ID 标识的人脸匹配的人脸的相关信息。

将 collectionID 的值更改为包含所需人脸的集合。将 faceId 的值更改为您要查找的人脸的标识符。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class SearchFacesMatchingId
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
```

```
        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        // Search collection for faces matching the face id.

        SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesResponse searchFacesResponse =
 rekognitionClient.SearchFaces(searchFacesRequest);

        Console.WriteLine("Face matching faceId " + faceId);

        Console.WriteLine("Matche(s): ");
        foreach (FaceMatch face in searchFacesResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

运行示例代码。将显示有关匹配的人脸的信息。

SearchFaces 操作请求

在给定人脸 ID（人脸集合中存储的每张人脸均有一个人脸 ID）的情况下，SearchFaces 将在指定人脸集合中搜索相似的人脸。该响应不包含您搜索的人脸。它仅包括相似的人脸。默认情况下，对于 SearchFaces 返回的人脸，算法检测到相似度得分高于 80%。相似度指示人脸与输入人脸的匹配程度。（可选）您可以使用 FaceMatchThreshold 指定不同的值。

```
{
    "CollectionId": "MyCollection",
    "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFaces 操作响应

此操作将返回一组找到的匹配人脸以及您作为输入提供的人脸 ID。

```
{
    "SearchedFaceId": "7ecf8c19-5274-5917-9c91-1db9ae0449e2",
    "FaceMatches": [ list of face matches found ]
}
```

对于找到的每个匹配的人脸，此响应将包含相似度和人脸元数据，如以下示例响应所示：

```
{
   ...
    "FaceMatches": [
        {
            "Similarity": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "FaceId": "84de1c86-5059-53f2-a432-34ebb704615d",
                "Confidence": 99.9997,
                "ImageId": "d38ebf91-1a11-58fc-ba42-f978b3f32f60"
            }
        },
        {
            "Similarity": 84.6859,
            "Face": {
                "BoundingBox": {
                    "Width": 0.2044,
                    "Top": 0.2254,
                    "Left": 0.4622,
                    "Height": 0.3119
                },
                "FaceId": "6fc892c7-5739-50da-a0d7-80cc92c0ba54",
                "Confidence": 99.9981,
                "ImageId": "5d913eaf-cf7f-5e09-8c8f-cb1bdea8e6aa"
            }
        }
    ]
```

```
}
```

使用图像搜索人脸

您可以使用 SearchFacesByImage 操作搜索集合中与所提供图像中最大的人脸匹配的人脸。

有关更多信息，请参阅 搜索集合内的人脸和用户。

使用图像搜索集合中的人脸 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将包含一张或多张人脸的图像上传到您的 S3 存储桶。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 SearchFacesByImage 操作。

   Java

   此示例显示与图像中的最大人脸匹配的人脸的相关信息。此代码示例同时指定 FaceMatchThreshold 和 MaxFaces 参数以限制响应中返回的结果。

   在以下示例中，更改以下内容：将 collectionId 的值更改为您想要搜索的集合，将 bucket 的值更改为包含输入图像的存储桶，将 photo 的值更改为输入图像。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
```

```
import com.amazonaws.services.rekognition.model.SearchFacesByImageRequest;
import com.amazonaws.services.rekognition.model.SearchFacesByImageResult;
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;


public class SearchFaceMatchingImageCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

    public static void main(String[] args) throws Exception {

       AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

       // Get an image object from S3 bucket.
      Image image=new Image()
              .withS3Object(new S3Object()
                      .withBucket(bucket)
                      .withName(photo));

      // Search collection for faces similar to the largest face in the image.
      SearchFacesByImageRequest searchFacesByImageRequest = new
 SearchFacesByImageRequest()
              .withCollectionId(collectionId)
              .withImage(image)
              .withFaceMatchThreshold(70F)
              .withMaxFaces(2);

       SearchFacesByImageResult searchFacesByImageResult =
               rekognitionClient.searchFacesByImage(searchFacesByImageRequest);

       System.out.println("Faces matching largest face in image from" + photo);
      List < FaceMatch > faceImageMatches =
 searchFacesByImageResult.getFaceMatches();
      for (FaceMatch face: faceImageMatches) {
          System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
                  .writeValueAsString(face));
         System.out.println();
      }
   }
```

```
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
// snippet-start:[rekognition.java2.search_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
// snippet-end:[rekognition.java2.search_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingImageCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
```

```
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection.  \n" +
            "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.search_faces_collection.main]
    public static void searchFaceInCollection(RekognitionClient rekClient,String
 collectionId, String sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(new
 File(sourceImage));
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            SearchFacesByImageRequest facesByImageRequest =
 SearchFacesByImageRequest.builder()
                .image(souImage)
                .maxFaces(10)
                .faceMatchThreshold(70F)
                .collectionId(collectionId)
                .build();
```

```
            SearchFacesByImageResponse imageResponse =
 rekClient.searchFacesByImage(facesByImageRequest) ;
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face: faceImageMatches) {
                System.out.println("The similarity level is
 "+face.similarity());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
    // snippet-end:[rekognition.java2.search_faces_collection.main]
}
```

AWS CLI

此 AWS CLI 命令显示 `search-faces-by-image` CLI 操作的 JSON 输出。将 `Bucket` 的值替换为您在步骤 2 中使用的 S3 存储桶。将 `Name` 的值替换为您在步骤 2 中使用的图像文件名。将 `collection-id` 的值替换为您要在其中执行搜索的集合。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
aws rekognition search-faces-by-image --image '{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}' \
--collection-id "collection-id" --profile profile-name
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition search-faces-by-image --image "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --profile profile-name
```

Python

此示例显示与图像中的最大人脸匹配的人脸的相关信息。此代码示例同时指定 FaceMatchThreshold 和 MaxFaces 参数以限制响应中返回的结果。

在以下示例中，更改以下内容：将 collectionId 的值更改为您想要搜索的集合，将bucket和photo的值分别替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='amzn-s3-demo-bucket'
    collectionId='MyCollection'
    fileName='input.jpg'
    threshold = 70
    maxFaces=2

    client=boto3.client('rekognition')


    response=client.search_faces_by_image(CollectionId=collectionId,
                                Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},
                                FaceMatchThreshold=threshold,
                                MaxFaces=maxFaces)


    faceMatches=response['FaceMatches']
    print ('Matching faces')
    for match in faceMatches:
            print ('FaceId:' + match['Face']['FaceId'])
            print ('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
            print
```

.NET

此示例显示与图像中的最大人脸匹配的人脸的相关信息。此代码示例同时指定 FaceMatchThreshold 和 MaxFaces 参数以限制响应中返回的结果。

在以下示例中，更改以下内容：将 collectionId 的值更改为您想要搜索的集合，将bucket和photo的值分别替换为您在步骤 2 中使用的 Amazon S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class SearchFacesMatchingImage
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "amzn-s3-demo-bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        SearchFacesByImageRequest searchFacesByImageRequest = new
 SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
            Image = image,
```

```
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesByImageResponse searchFacesByImageResponse =
 rekognitionClient.SearchFacesByImage(searchFacesByImageRequest);

        Console.WriteLine("Faces matching largest face in image from " + photo);
        foreach (FaceMatch face in searchFacesByImageResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

SearchFacesByImage 操作请求

SearchFacesImageByImage 的输入参数是要在其中进行搜索的集合和源图像位置。在此示例中，源图像存储在 Amazon S3 存储桶（S3Object）中。另外还指定了要返回的人脸最大数量 (Maxfaces) 和要与返回的人脸匹配所必须达到的最低置信度 (FaceMatchThreshold)。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFacesByImage 操作响应

在给定输入图像（.jpeg 或 .png）的情况下，此操作会先检测输入图像中的人脸，然后在指定的人脸集合中搜索相似的人脸。

> **Note**
>
> 如果服务在输入图像中检测到多个人脸，它会使用检测到的最大人脸来搜索人脸集合。

此操作将返回一组找到的匹配人脸以及有关输入人脸的信息。这包括如下信息：边界框以及置信度值，后者指示边界框包含人脸的置信度级别。

默认情况下，对于 SearchFacesByImage 返回的人脸，算法检测到相似度得分高于 80%。相似度指示人脸与输入人脸的匹配程度。（可选）您可以使用 FaceMatchThreshold 指定不同的值。对于找到的每个匹配的人脸，此响应将包含相似度和人脸元数据，如以下示例响应所示：


```
{
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333330273628235,
                    "Left": 0.1718519926071167,
                    "Top": 0.7366669774055481,
                    "Width": 0.11061699688434601
                },
                "Confidence": 100,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "Similarity": 99.9764175415039
        }
    ],
    "FaceModelVersion": "3.0",
    "SearchedFaceBoundingBox": {
        "Height": 0.06333333253860474,
        "Left": 0.17185185849666595,
        "Top": 0.7366666793823242,
        "Width": 0.11061728745698929
    },
    "SearchedFaceConfidence": 99.99999237060547
}
```


搜索用户（人脸 ID /用户 ID）

您可以使用该SearchUsers操作在指定集合中搜索与提供的面容 ID 或用户 ID 相匹配的用户。该操作列出了按高于请求 UserMatchThreshold值的最高相似度分数UserIds排名的返回结果。用户 ID 是在 CreateUsers 操作中创建的。有关更多信息，请参阅 在集合中管理用户。

搜索用户 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 SearchUsers 操作。

   Java

   此 Java 示例使用 SearchUsers 操作搜索集合中的用户。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.UserMatch;
import com.amazonaws.services.rekognition.model.SearchUsersRequest;
import com.amazonaws.services.rekognition.model.SearchUsersResult;
import com.amazonaws.services.rekognition.model.UserMatch;

public class SearchUsers {
    //Replace collectionId and faceId with the values you want to use.

    public static final String collectionId = "MyCollection";
    public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

    public static final String userd = 'demo-user';

    public static void main(String[] args) throws Exception {

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        // Search collection for faces matching the user id.
        SearchUsersRequest request = new SearchUsersRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        SearchUsersResult result =
                rekognitionClient.searchUsers(request);
```

```
        System.out.println("Printing first search result with matched user and
similarity score");
        for (UserMatch match: result.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
        }

        // Search collection for faces matching the face id.
        SearchUsersRequest request1 = new SearchUsersRequest()
                .withCollectionId(collectionId)
                .withFaceId(faceId);

        SearchUsersResult result1 =
                rekognitionClient.searchUsers(request1);

        System.out.println("Printing second search result with matched user and
similarity score");
        for (UserMatch match: result1.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
        }
    }
```

AWS CLI

此 AWS CLI 命令使用SearchUsers操作搜索集合中的用户。

```
aws rekognition search-users --face-id face-id --collection-id collection-id --
region region-name
```

Python

以下示例使用 SearchUsers 操作搜索集合中的用户。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
import logging

logger = logging.getLogger(__name__)
```

```
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def search_users_by_face_id(collection_id, face_id):
    """
    SearchUsers operation with face ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param face_id: The ID of the face in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a face-id: {face_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
            FaceId=face_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {face_id}')
        raise
    else:
        print(response)
        return response

def search_users_by_user_id(collection_id, user_id):
    """
    SearchUsers operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a user-id: {user_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
```

```
            UserId=user_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {user_id}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    face_id = "face_id"
    search_users_by_face_id(collection_id, face_id)
    search_users_by_user_id(collection_id, user_id)

if __name__ == "__main__":
    main()
```

SearchUsers 操作请求

给定 FaceID 或用户 ID SearchUsers ，在指定的 CollectionId 中搜索用户匹配项。默认情况下，SearchUsers 返回相似度分数大于 80% 的用户IDs 。相似度表示用户 ID 与提供的 FaceID 或用户 ID 的匹配程度。如果返回多个用户IDs ，则按相似度分数最高到最低的顺序列出。或者，您可以使用 UserMatchThreshold 来指定不同的值。有关更多信息，请参阅 在集合中管理用户。

以下是使用以下 SearchUsers 请求的示例`UserId`：

```
{
   "CollectionId": "MyCollection",
   "UserId": "demoUser1",
   "MaxUsers": 2,
   "UserMatchThreshold": 99
}
```

以下是使用以下 SearchUsers 请求的示例FaceId：

```
{
    "CollectionId": "MyCollection",
    "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107",
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsers 操作响应

如果使用 a 进行搜索FaceId，则对FaceId的响应将 SearchUsers SearchedFace包括每个用户的列表以及UserMatchesUserId和UserStatus的列表。

```
{
    "SearchedFace": {
        "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

如果使用 a 进行搜索UserId，则除了其他响应元素外SearchedUser，的响应还 SearchUsers 包括对的。UserId

```
{
    "SearchedUser": {
        "UserId": "demoUser1"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

搜索用户（图像）

SearchUsersByImage 在指定集合 ID 中搜索与所提供图像中检测到的最大人脸相匹配的用户。默认情况下， SearchUsersByImage 返回相似度分数大于 80% 的用户IDs 。相似度表示用户 ID 与所提供图像中检测到的最大面孔的匹配程度。如果返回多个用户IDs ，则按相似度分数最高到最低的顺序列出。或者，您可以使用 UserMatchThreshold 来指定不同的值。有关更多信息，请参阅 the section called “在集合中管理用户”。

按图片搜索用户 (SDK)

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下示例调用 SearchUsersByImage 操作。

Java

此 Java 示例使用 SearchUsersByImage 操作根据输入图像搜索集合中的用户。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.SearchUsersByImageRequest;
import com.amazonaws.services.rekognition.model.SearchUsersByImageResult;
import com.amazonaws.services.rekognition.model.UserMatch;


public class SearchUsersByImage {
    //Replace bucket, collectionId and photo with your values.
    public static final String collectionId = "MyCollection";
    public static final String s3Bucket = "bucket";
    public static final String s3PhotoFileKey = "input.jpg";

    public static void main(String[] args) throws Exception {

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


        // Get an image object from S3 bucket.
        Image image = new Image()
                .withS3Object(new S3Object()
                        .withBucket(s3Bucket)
                        .withName(s3PhotoFileKey));

        // Search collection for users similar to the largest face in the image.
        SearchUsersByImageRequest request = new SearchUsersByImageRequest()
                .withCollectionId(collectionId)
                .withImage(image)
                .withUserMatchThreshold(70F)
                .withMaxUsers(2);

        SearchUsersByImageResult result =
                rekognitionClient.searchUsersByImage(request);

        System.out.println("Printing search result with matched user and
 similarity score");
```

```
        for (UserMatch match: result.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
 score " + match.getSimilarity());
        }
    }
}
```

AWS CLI

此 AWS CLI 命令通过SearchUsersByImage操作根据输入图像在集合中搜索用户。

```
aws rekognition search-users-by-image --image '{"S3Object":
{"Bucket":"s3BucketName","Name":"file-name"}}' --collection-id MyCollectionId --
region region-name
```

Python

以下示例使用 SearchUsersByImage 操作根据输入图像搜索集合中的用户。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
import logging
import os

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def load_image(file_name):
    """
    helper function to load the image for indexFaces call from local disk

    :param image_file_name: The image file location that will be used by
 indexFaces call.
    :return: The Image in bytes
    """
    print(f'- loading image: {file_name}')
    with open(file_name, 'rb') as file:
        return {'Bytes': file.read()}
```

```
def search_users_by_image(collection_id, image_file):
    """
    SearchUsersByImage operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param image_file: The image that contains the reference face to search
 for.

    :return: response of SearchUsersByImage API
    """
    logger.info(f'Searching for users using an image: {image_file}')
    try:
        response = client.search_users_by_image(
            CollectionId=collection_id,
            Image=load_image(image_file)
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsersByImage with given
 image: {image_file}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    IMAGE_SEARCH_SOURCE = os.getcwd() + '/image_path'
    search_users_by_image(collection_id, IMAGE_SEARCH_SOURCE)

if __name__ == "__main__":
    main()
```

SearchUsersByImage 操作请求

对 SearchUsersByImage 的请求包括要在其中进行搜索的集合和源图像位置。在此示例中，源图像存储在 Amazon S3 存储桶（S3Object）中。另外还指定了要返回的用户最大数量（MaxUsers）和要与返回的用户匹配所必须达到的最低置信度（UserMatchThreshold）。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsersByImage 操作响应

的响应SearchUsersByImage包括一个对应的FaceDetail对象SearchedFace，以及每个对象 UserMatches 的列表，均UserStatus为UserIdSimilarity、和。如果输入图像包含多张脸，则还 UnsearchedFaces 会返回一个列表。

```
{
    "SearchedFace": {
        "FaceDetail": {
            "BoundingBox": {
                "Width": 0.23692893981933594,
                "Top": 0.19235000014305115,
                "Left": 0.39177176356315613,
                "Height": 0.5437348484992981
            }
        }
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
```

```
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6",
    "UnsearchedFaces": [
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.031677018851041794,
                    "Top": 0.5593535900115967,
                    "Left": 0.6102562546730042,
                    "Height": 0.0682177022099495
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        },
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.03254449740052223,
                    "Top": 0.6080358028411865,
                    "Left": 0.516062319278717,
                    "Height": 0.06347997486591339
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        }
    ]
}
```


搜索存储视频中的人脸

您可在集合中搜索与存储视频或流视频中检测到的人脸匹配的人脸。此节介绍搜索存储视频中的人脸。有关搜索流式传输视频中的人脸的信息，请参阅使用流视频事件。

必须先通过使用 IndexFaces 将您搜索的人脸索引到集合中。有关更多信息，请参阅 将人脸添加到集合。

Amazon Rekognition Video 人脸搜索将与其他分析 Amazon S3 存储桶中所存储视频的 Amazon Rekognition Video 操作执行相同的异步工作流程。要开始在存储的视频中搜索面孔，请致电StartFaceSearch并提供您要搜索的集合的 ID。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service (Amazon SNS) 主题。如果视频分析成功，请调用 GetFaceSearch 来获取搜索结果。有关启动视频分析和获取结果的详细信息，请参阅调用 Amazon Rekognition Video 操作。

以下过程演示如何在集合中搜索与视频中检测到的人脸匹配的人脸。此过程还演示了如何获取视频中匹配人员的跟踪数据。此过程在使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon Simple Queue Service (Amazon SQS) 队列获取视频分析请求的完成状态）中的代码的基础上进行了扩展。

在视频中搜索匹配的人脸 (SDK)

1. 创建集合。
2. 将人脸索引到集合中。
3. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
4. 将以下代码添加到您在步骤 3 中创建的类 `VideoDetect`。

   Java

```
   //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
   //PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

       //Face collection search in video
 ======================================================================
       private static void StartFaceSearchCollection(String bucket, String
 video, String collection) throws Exception{

        NotificationChannel channel= new NotificationChannel()
                .withSNSTopicArn(snsTopicArn)
                .withRoleArn(roleArn);

        StartFaceSearchRequest req = new StartFaceSearchRequest()
                .withCollectionId(collection)
                .withVideo(new Video()
                        .withS3Object(new S3Object()
```

```
                                .withBucket(bucket)
                                .withName(video)))
                .withNotificationChannel(channel);



       StartFaceSearchResult startPersonCollectionSearchResult =
rek.startFaceSearch(req);
       startJobId=startPersonCollectionSearchResult.getJobId();

   }

   //Face collection search in video
======================================================================
   private static void GetFaceSearchCollectionResults() throws Exception{

      GetFaceSearchResult faceSearchResult=null;
      int maxResults=10;
      String paginationToken=null;

      do {

          if (faceSearchResult !=null){
              paginationToken = faceSearchResult.getNextToken();
          }


          faceSearchResult  = rek.getFaceSearch(
                  new GetFaceSearchRequest()
                  .withJobId(startJobId)
                  .withMaxResults(maxResults)
                  .withNextToken(paginationToken)
                  .withSortBy(FaceSearchSortBy.TIMESTAMP)
                  );


          VideoMetadata videoMetaData=faceSearchResult.getVideoMetadata();

          System.out.println("Format: " + videoMetaData.getFormat());
          System.out.println("Codec: " + videoMetaData.getCodec());
          System.out.println("Duration: " +
videoMetaData.getDurationMillis());
          System.out.println("FrameRate: " + videoMetaData.getFrameRate());
          System.out.println();
```

```
          //Show search results
          List<PersonMatch> matches=
                  faceSearchResult.getPersons();

          for (PersonMatch match: matches) {
              long milliSeconds=match.getTimestamp();
              System.out.print("Timestamp: " + Long.toString(milliSeconds));
              System.out.println(" Person number: " +
match.getPerson().getIndex());
              List <FaceMatch> faceMatches = match.getFaceMatches();
              if (faceMatches != null) {
                  System.out.println("Matches in collection...");
                  for (FaceMatch faceMatch: faceMatches){
                      Face face=faceMatch.getFace();
                      System.out.println("Face Id: "+ face.getFaceId());
                      System.out.println("Similarity: " +
faceMatch.getSimilarity().toString());
                      System.out.println();
                  }
              }
              System.out.println();
          }

          System.out.println();

      } while (faceSearchResult !=null && faceSearchResult.getNextToken() !=
null);

  }
```

在函数 `main` 中，将以下行:

```
       StartLabelDetection(bucket, video);

       if (GetSQSMessageSuccess()==true)
        GetLabelDetectionResults();
```

替换为:

```
       String collection="collection";
```

```
        StartFaceSearchCollection(bucket, video, collection);

        if (GetSQSMessageSuccess()==true)
         GetFaceSearchCollectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectFaces {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startFaceDetection(rekClient, channel, bucket, video);
        getFaceResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startFaceDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartFaceDetectionRequest faceDetectionRequest =
StartFaceDetectionRequest.builder()
                    .jobTag("Faces")
                    .faceAttributes(FaceAttributes.ALL)
                    .notificationChannel(channel)
                    .video(vidOb)
```

```
                    .build();

            StartFaceDetectionResponse startLabelDetectionResult =
rekClient.startFaceDetection(faceDetectionRequest);
            startJobId = startLabelDetectionResult.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getFaceResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetFaceDetectionResponse faceDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (faceDetectionResponse != null)
                    paginationToken = faceDetectionResponse.nextToken();

                GetFaceDetectionRequest recognitionRequest =
GetFaceDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {

                    faceDetectionResponse =
rekClient.getFaceDetection(recognitionRequest);
                    status = faceDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
```

```
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 faceDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                // Show face information.
                List<FaceDetection> faces = faceDetectionResponse.faces();
                for (FaceDetection face : faces) {
                    String age = face.face().ageRange().toString();
                    String smile = face.face().smile().toString();
                    System.out.println("The detected face is estimated to be"
                            + age + " years old.");
                    System.out.println("There is a smile : " + smile);
                }

            } while (faceDetectionResponse != null &&
 faceDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Face Search ===============
```

```
    def StartFaceSearchCollection(self,collection):
        response = self.rek.start_face_search(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            CollectionId=collection,
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.snsTopicArn})

        self.startJobId=response['JobId']

        print('Start Job Id: ' + self.startJobId)


    def GetFaceSearchCollectionResults(self):
        maxResults = 10
        paginationToken = ''

        finished = False

        while finished == False:
            response = self.rek.get_face_search(JobId=self.startJobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for personMatch in response['Persons']:
                print('Person Index: ' + str(personMatch['Person']['Index']))
                print('Timestamp: ' + str(personMatch['Timestamp']))

                if ('FaceMatches' in personMatch):
                    for faceMatch in personMatch['FaceMatches']:
                        print('Face ID: ' + faceMatch['Face']['FaceId'])
                        print('Similarity: ' + str(faceMatch['Similarity']))
                print()
            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
            print()
```

在函数 main 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    collection='tests'
    analyzer.StartFaceSearchCollection(collection)

    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceSearchCollectionResults()
```

如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

5. 将 collection 的值更改为您在步骤 1 中创建的集合的名称。
6. 运行该代码。将显示视频中人脸与输入集合中的人脸匹配的人员的列表。还将显示每个匹配人员的跟踪数据。

GetFaceSearch 操作响应

以下是来自 GetFaceSearch 的示例 JSON 响应。

该响应包括在视频中检测到的其人脸与输入集合中的人脸匹配的一组人员 (Persons)。每当在视频中匹配到该人员时，都会存在一个 PersonMatch 对象。每个 PersonMatch 都包含输入集合中的人脸匹配的数组 FaceMatch、有关匹配人员的信息 PersonDetail 以及在视频中匹配到人员的时间。

```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "IJdbzkZfvBRqj8GPV82BPiZKkLOGCqDIsNZG/gQsEE5faTVK9JHOz/
xxxxxxxxxxxxxx",
    "Persons": [
        {
            "FaceMatches": [
                {
                    "Face": {
                        "BoundingBox": {
```

```
                        "Height": 0.527472972869873,
                        "Left": 0.33530598878860474,
                        "Top": 0.2161169946193695,
                        "Width": 0.35503000020980835
                    },
                    "Confidence": 99.90239715576172,
                    "ExternalImageId": "image.PNG",
                    "FaceId": "a2f2e224-bfaa-456c-b360-7c00241e5e2d",
                    "ImageId": "eb57ed44-8d8d-5ec5-90b8-6d190daff4c3"
                },
                "Similarity": 98.40909576416016
            }
        ],
        "Person": {
            "BoundingBox": {
                "Height": 0.8694444298744202,
                "Left": 0.2473958283662796,
                "Top": 0.10092592239379883,
                "Width": 0.49427083134651184
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
                "Confidence": 99.97504425048828,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.46415066719055176,
                        "Y": 0.2572723925113678
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.5068183541297913,
                        "Y": 0.23705792427062988
                    },
                    {
                        "Type": "nose",
                        "X": 0.49765899777412415,
                        "Y": 0.28383663296699524
                    },
```

```
                    {
                        "Type": "mouthLeft",
                        "X": 0.487221896648407,
                        "Y": 0.3452930748462677
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.5142884850502014,
                        "Y": 0.33167609572410583
                    }
                ],
                "Pose": {
                    "Pitch": 15.966927528381348,
                    "Roll": -15.547388076782227,
                    "Yaw": 11.34195613861084
                },
                "Quality": {
                    "Brightness": 44.80223083496094,
                    "Sharpness": 99.95819854736328
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.2177777737379074,
                "Left": 0.7593749761581421,
                "Top": 0.13333334028720856,
                "Width": 0.12250000238418579
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.2177777737379074,
                    "Left": 0.7593749761581421,
                    "Top": 0.13333334028720856,
                    "Width": 0.12250000238418579
                },
                "Confidence": 99.63436889648438,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
```

```
                        "X": 0.8005779385566711,
                        "Y": 0.20915353298187256
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.8391435146331787,
                        "Y": 0.21049551665782928
                    },
                    {
                        "Type": "nose",
                        "X": 0.8191410899162292,
                        "Y": 0.2523227035999298
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.8093273043632507,
                        "Y": 0.29053622484207153
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.8366993069648743,
                        "Y": 0.29101791977882385
                    }
                ],
                "Pose": {
                    "Pitch": 3.165884017944336,
                    "Roll": 1.4182015657424927,
                    "Yaw": -11.151537895202637
                },
                "Quality": {
                    "Brightness": 28.910892486572266,
                    "Sharpness": 97.61507415771484
                }
            },
            "Index": 1
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.8388888835906982,
                "Left": 0,
                "Top": 0.15833333134651184,
```

```
                    "Width": 0.2369791716337204
                },
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20000000298023224,
                        "Left": 0.029999999329447746,
                        "Top": 0.2199999988079071,
                        "Width": 0.11249999701976776
                    },
                    "Confidence": 99.85971069335938,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.06842322647571564,
                            "Y": 0.3010137975215912
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.10543643683195114,
                            "Y": 0.29697132110595703
                        },
                        {
                            "Type": "nose",
                            "X": 0.09569807350635529,
                            "Y": 0.33701086044311523
                        },
                        {
                            "Type": "mouthLeft",
                            "X": 0.0732642263174057,
                            "Y": 0.3757539987564087
                        },
                        {
                            "Type": "mouthRight",
                            "X": 0.10589495301246643,
                            "Y": 0.3722417950630188
                        }
                    ],
                    "Pose": {
                        "Pitch": -0.5589138865470886,
                        "Roll": -5.1093974113464355,
                        "Yaw": 18.69594955444336
                    },
                    "Quality": {
                        "Brightness": 43.052337646484375,
```

```
                        "Sharpness": 99.68138885498047
                    }
                },
                "Index": 2
            },
            "Timestamp": 0
        }......

    ],
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

在流视频中搜索集合中的人脸

您可以使用 Amazon Rekognition Video 在流视频中检测和识别集合中的人脸。使用 Amazon Rekognition Video，你可以创建一个流处理器 CreateStreamProcessor() 来启动和管理流媒体视频的分析。

为检测视频流中的已知人脸（人脸搜索），Amazon Rekognition Video 使用 Amazon Kinesis Video Streams 接收和处理视频流。分析结果将从 Amazon Rekognition Video 输出到 Kinesis 数据流，然后由您的客户端应用程序进行读取。

要将 Amazon Rekognition Video 与流视频结合使用，您的应用程序需要实施以下内容：

- 用于向 Amazon Rekognition Video 发送流视频的 Kinesis 视频流。有关更多信息，请参阅 Amazon Kinesis Video Streams 开发人员指南。
- 一个 Amazon Rekognition Video 流处理器，用于管理对流视频的分析。有关更多信息，请参阅 Amazon Rekognition Video 流处理器操作概述。
- Kinesis 数据流使用者，用于读取 Amazon Rekognition Video 发送到 Kinesis 数据流的分析结果。有关更多信息，请参阅 Kinesis 数据流使用者。

本节包含有关编写用于创建 Kinesis 视频流和其他必要资源、将视频流式传输到 Amazon Rekognition Video 以及接收分析结果的应用程序的信息。

主题

- 设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源
- 在流视频中搜索人脸
- 使用 GStreamer插件进行直播
- 流视频问题排查

设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源

以下过程描述了配置 Kinesis 视频流和用于识别流视频中人脸的其他资源的步骤。

先决条件

要运行此过程，你需要 适用于 Java 的 AWS SDK 安装。有关更多信息，请参阅 Amazon Rekognition 入门。 AWS 账户 您使用的必须拥有亚马逊 Rekognition API 的访问权限。有关更多信息，请参阅 IAM 用户指南中的 Amazon Rekognition 定义的操作。

识别视频流中的人脸 (AWS SDK)

1. 如果您还没有这样做，请创建一个 IAM 服务角色，让 Amazon Rekognition Video 有权访问您的 Kinesis 视频流和 Kinesis 数据流。记下 ARN。有关更多信息，请参阅 使用授予直播访问权限 AmazonRekognitionServiceRole。
2. 创建集合并记下您使用的集合标识符。
3. 将要搜索的人脸索引到您在步骤 2 中创建的集合中。
4. 创建 Kinesis 视频流并记下流的 Amazon 资源名称 (ARN)。
5. 创建 Kinesis 数据流。在直播名称前加上AmazonRekognition并记下直播的 ARN。

然后，您可以使用所选的流处理器名称创建人脸搜索流处理器并启动流处理器。

> **Note**
>
> 只有在确认可以将媒体摄取到 Kinesis 视频流之后，才应启动流处理器。

将视频流式传输到 Amazon Rekognition Video 中

要将视频流式传输到 Amazon Rekognition Video，您可以使用 Amazon Kinesis Video Streams SDK 来创建和使用 Kinesis 视频流。`PutMedia` 操作会将视频数据片段写入到 Amazon Rekognition

Video 使用的 Kinesis 视频流中。每个视频数据片段的时长通常为 2-10 秒，并且包含一系列独立视频帧。Amazon Rekognition Video 支持 H.264 编码的视频，该视频可以有三种类型的帧（I、B 和 P）。有关更多信息，请参阅帧间。版本中的第一个帧必须为 I 帧。I 帧可独立于任何其他帧进行解码。

当视频数据进入 Kinesis 视频流时，Kinesis 视频流会向片段分配一个唯一编号。有关示例，请参阅 PutMedia API 示例。

- 如果您从 Matroska (MKV) 编码的源进行流式传输，请使用PutMedia操作将源视频流式传输到您创建的 Kinesis 视频流中。有关更多信息，请参阅 PutMedia API 示例。
- 如果您从设备摄像头进行流式传输，请参阅使用 GStreamer插件进行直播。

允许 Amazon Rekognition Video 访问您的资源

您可以使用 AWS Identity and Access Management (IAM) 服务角色向亚马逊 Rekognition Video 授予 Kinesis 视频流的读取权限。如果您使用的是人脸搜索流处理器，则可以使用 IAM 服务角色向 Amazon Rekognition Video 授予对 Kinesis 数据流的写入权限。如果您使用的是安全监控流处理器，则可以使用 IAM 角色授予 Amazon Rekognition Video 访问您的 Amazon S3 存储桶和 Amazon SNS 主题的权限。

为人脸搜索流处理器提供访问权限

您可以创建权限策略，允许 Amazon Rekognition Video 访问单个 Kinesis 视频流和 Kinesis 数据流。

允许 Amazon Rekognition Video 访问人脸搜索流处理器

1. 使用 IAM JSON 策略编辑器创建新的权限策略，然后使用以下策略。将 `video-arn` 替换为所需的 Kinesis 视频流的 ARN。如果您使用的是人脸搜索流处理器，请将 `data-arn` 替换为所需的 Kinesis 数据流的 ARN。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "data-arn"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "video-arn"
        }
    ]
}
```

2. 创建 IAM 服务角色，或者更新现有 IAM 服务角色。使用以下信息创建 IAM 服务角色：

    1. 对于服务名称，选择 Rekognition。
    2. 对于服务角色使用案例，选择 Rekognition。
    3. 附加您在步骤 1 中创建的权限策略。
3. 记下服务角色的 ARN。您需要它才能开始视频分析操作。

使用授予直播访问权限 AmazonRekognitionServiceRole

作为设置 Kinesis 视频流和数据流访问权限的替代选项，您可以使用 `AmazonRekognitionServiceRole` 权限策略。IAM 提供 Rekognition 服务角色使用案例，当与 `AmazonRekognitionServiceRole` 权限策略结合使用时，该案例可以写入多个 Kinesis 数据流并从您的所有 Kinesis 数据流中进行读取。要向 Amazon Rekognition Video 授予对多个 Kinesis 数据流的写入权限，你可以在 Kinesis 数据流的名称前面加上——例如。AmazonRekognition`AmazonRekognitionMyDataStreamName`

允许 Amazon Rekognition Video 访问您的 Kinesis 视频流和 Kinesis 数据流

1. 创建 IAM 服务角色。使用以下信息创建 IAM 服务角色：

    1. 对于服务名称，选择 Rekognition。
    2. 对于服务角色使用案例，选择 Rekognition。
    3. 选择AmazonRekognitionServiceRole权限策略，该策略授予亚马逊 Rekognition Video 对以您的所有 Kinesis 视频流为前缀的 Kinesis 数据流的写入权限，AmazonRekognition以及对所有 Kinesis 视频流的读取权限。
2. 为确保您的安全，请将 Rekognition 的访问范围限制为仅限于您 AWS 账户 正在使用的资源。这可以通过将信任策略附加到您的 IAM 服务角色来完成。有关如何执行此操作的信息，请参阅 防止跨服务混淆代理。

3. 记下服务角色的 Amazon 资源名称 (ARN)。您需要它才能开始视频分析操作。

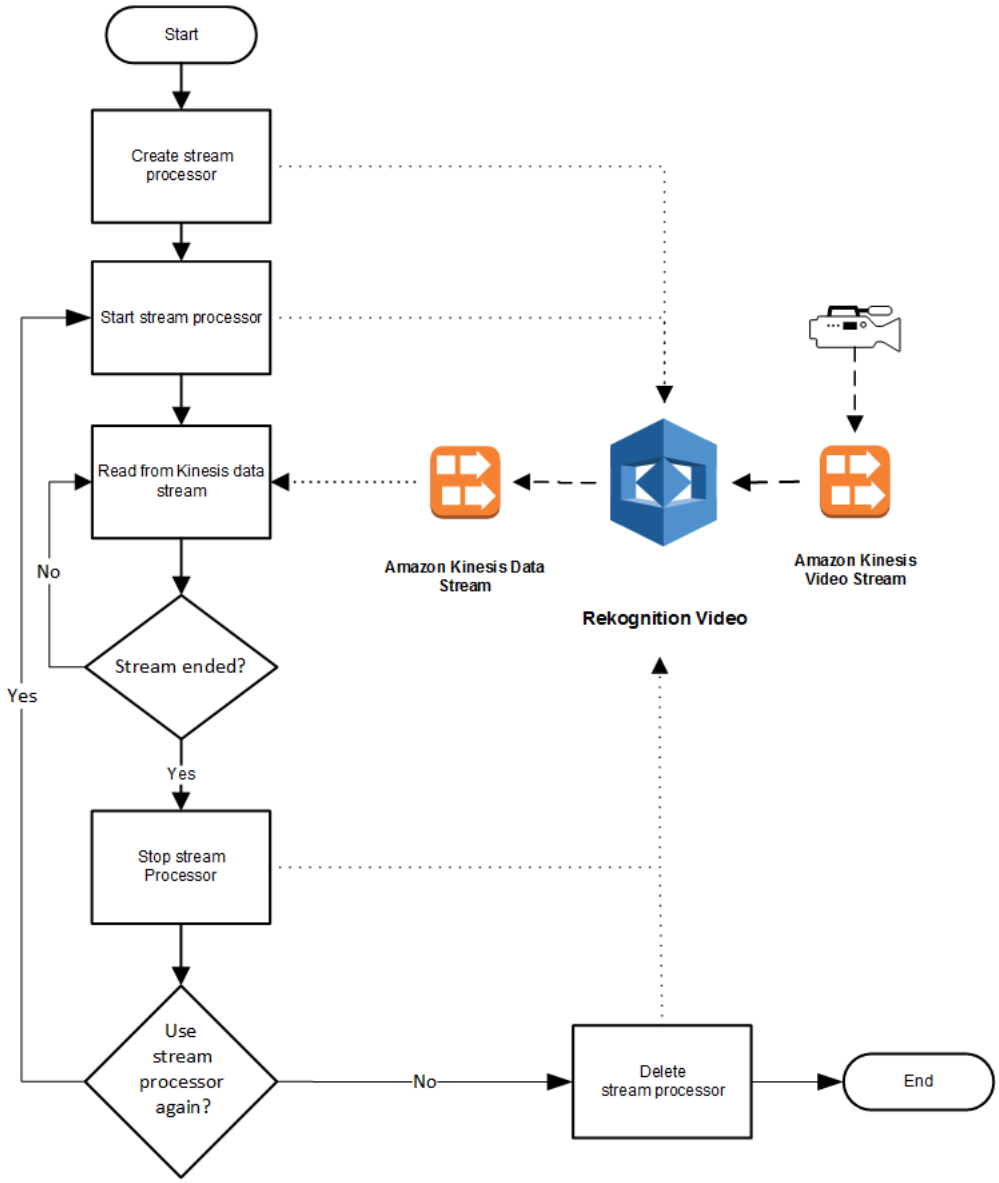
在流视频中搜索人脸

Amazon Rekognition Video 可以搜索集合中与在流视频中检测到的人脸匹配的人脸。有关集合的更多信息，请参阅在集合中搜索人脸。

主题

- 创建 Amazon Rekognition Video 人脸搜索流处理器
- 启动 Amazon Rekognition Video 人脸搜索流处理器
- 使用流处理器搜索人脸（Java V2 示例）
- 使用流处理器搜索人脸（Java V1 示例）
- 读取流视频分析结果
- 在本地使用 Kinesis 视频流显示 Rekognition 结果
- 了解 Kinesis 人脸识别 JSON 帧记录

下图显示了 Amazon Rekognition Video 如何检测和识别流视频中的人脸。



创建 Amazon Rekognition Video 人脸搜索流处理器

在分析流媒体视频之前，您需要先创建一个 Amazon Rekognition Video 流处理器 ()。CreateStreamProcessor流处理器包含有关 Kinesis 数据流和 Kinesis 视频流的信息。它还包含含有您要在输入流视频中识别的人脸的集合的标识符。您还可为流处理器指定名称。以下是 `CreateStreamProcessor` 请求的 JSON 示例。

```
{
	"Name": "streamProcessorForCam",
	"Input": {
		"KinesisVideoStream": {
```

```
                    "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/
inputVideo"
                }
        },
        "Output": {
                "KinesisDataStream": {
                        "Arn": "arn:aws:kinesis:us-east-1:nnnnnnnnnnnn:stream/outputData"
                }
        },
        "RoleArn": "arn:aws:iam::nnnnnnnnnnn:role/roleWithKinesisPermission",
        "Settings": {
                "FaceSearch": {
                        "CollectionId": "collection-with-100-faces",
                        "FaceMatchThreshold": 85.5
                }
        }
}
```

以下是来自 CreateStreamProcessor 的示例响应。

```
{
        “StreamProcessorArn”: “arn:aws:rekognition:us-
east-1:nnnnnnnnnnnn:streamprocessor/streamProcessorForCam”
}
```

启动 Amazon Rekognition Video 人脸搜索流处理器

您可使用在 CreateStreamProcessor 中指定的流处理器名称来调用 StartStreamProcessor，由此开始分析流视频。以下是 StartStreamProcessor 请求的 JSON 示例。

```
{
        "Name": "streamProcessorForCam"
}
```

如果流处理器成功启动，则会返回 HTTP 200 响应以及空白的 JSON 正文。

使用流处理器搜索人脸（Java V2 示例）

以下示例代码展示了如何使用适用于 Java 的 AWS SDK 版本 2 调用各种流处理器操作 StartStreamProcessor，例如CreateStreamProcessor和。

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorRequest;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorResponse;
import software.amazon.awssdk.services.rekognition.model.FaceSearchSettings;
import software.amazon.awssdk.services.rekognition.model.KinesisDataStream;
import software.amazon.awssdk.services.rekognition.model.KinesisVideoStream;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsRequest;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.StreamProcessor;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorInput;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorSettings;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorOutput;
import software.amazon.awssdk.services.rekognition.model.StartStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CreateStreamProcessor {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <role> <kinInputStream> <kinOutputStream> <collectionName>
 <StreamProcessorName>

                Where:
                   role - The ARN of the AWS Identity and Access Management (IAM) role
 to use. \s
                   kinInputStream - The ARN of the Kinesis video stream.\s
                   kinOutputStream - The ARN of the Kinesis data stream.\s
                   collectionName - The name of the collection to use that contains
 content. \s
```

```
                    StreamProcessorName - The name of the Stream Processor. \s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String role = args[0];
        String kinInputStream = args[1];
        String kinOutputStream = args[2];
        String collectionName = args[3];
        String streamProcessorName = args[4];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        processCollection(rekClient, streamProcessorName, kinInputStream,
kinOutputStream, collectionName,
                role);
        startSpecificStreamProcessor(rekClient, streamProcessorName);
        listStreamProcessors(rekClient);
        describeStreamProcessor(rekClient, streamProcessorName);
        deleteSpecificStreamProcessor(rekClient, streamProcessorName);
    }

    public static void listStreamProcessors(RekognitionClient rekClient) {
        ListStreamProcessorsRequest request = ListStreamProcessorsRequest.builder()
                .maxResults(15)
                .build();

        ListStreamProcessorsResponse listStreamProcessorsResult =
rekClient.listStreamProcessors(request);
        for (StreamProcessor streamProcessor :
listStreamProcessorsResult.streamProcessors()) {
            System.out.println("StreamProcessor name - " + streamProcessor.name());
            System.out.println("Status - " + streamProcessor.status());
        }
    }

    private static void describeStreamProcessor(RekognitionClient rekClient, String
StreamProcessorName) {
```

```
        DescribeStreamProcessorRequest streamProcessorRequest =
DescribeStreamProcessorRequest.builder()
                .name(StreamProcessorName)
                .build();

        DescribeStreamProcessorResponse describeStreamProcessorResult = rekClient
                .describeStreamProcessor(streamProcessorRequest);
        System.out.println("Arn - " +
describeStreamProcessorResult.streamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
                + describeStreamProcessorResult.input().kinesisVideoStream().arn());
        System.out.println("Output kinesisData stream - "
                + describeStreamProcessorResult.output().kinesisDataStream().arn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.roleArn());
        System.out.println(
                "CollectionId - "
                        +
describeStreamProcessorResult.settings().faceSearch().collectionId());
        System.out.println("Status - " + describeStreamProcessorResult.status());
        System.out.println("Status message - " +
describeStreamProcessorResult.statusMessage());
        System.out.println("Creation timestamp - " +
describeStreamProcessorResult.creationTimestamp());
        System.out.println("Last update timestamp - " +
describeStreamProcessorResult.lastUpdateTimestamp());
    }

    private static void startSpecificStreamProcessor(RekognitionClient rekClient,
String StreamProcessorName) {
        try {
            StartStreamProcessorRequest streamProcessorRequest =
StartStreamProcessorRequest.builder()
                    .name(StreamProcessorName)
                    .build();

            rekClient.startStreamProcessor(streamProcessorRequest);
            System.out.println("Stream Processor " + StreamProcessorName + "
started.");

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
   private static void processCollection(RekognitionClient rekClient, String
StreamProcessorName,
                                         String kinInputStream, String
kinOutputStream, String collectionName, String role) {
       try {
           KinesisVideoStream videoStream = KinesisVideoStream.builder()
                   .arn(kinInputStream)
                   .build();

           KinesisDataStream dataStream = KinesisDataStream.builder()
                   .arn(kinOutputStream)
                   .build();

           StreamProcessorOutput processorOutput = StreamProcessorOutput.builder()
                   .kinesisDataStream(dataStream)
                   .build();

           StreamProcessorInput processorInput = StreamProcessorInput.builder()
                   .kinesisVideoStream(videoStream)
                   .build();

           FaceSearchSettings searchSettings = FaceSearchSettings.builder()
                   .faceMatchThreshold(75f)
                   .collectionId(collectionName)
                   .build();

           StreamProcessorSettings processorSettings =
StreamProcessorSettings.builder()
                   .faceSearch(searchSettings)
                   .build();

           CreateStreamProcessorRequest processorRequest =
CreateStreamProcessorRequest.builder()
                   .name(StreamProcessorName)
                   .input(processorInput)
                   .output(processorOutput)
                   .roleArn(role)
                   .settings(processorSettings)
                   .build();

           CreateStreamProcessorResponse response =
rekClient.createStreamProcessor(processorRequest);
           System.out.println("The ARN for the newly create stream processor is "
```

```
                    + response.streamProcessorArn());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    private static void deleteSpecificStreamProcessor(RekognitionClient rekClient,
 String StreamProcessorName) {
        rekClient.stopStreamProcessor(a -> a.name(StreamProcessorName));
        rekClient.deleteStreamProcessor(a -> a.name(StreamProcessorName));
        System.out.println("Stream Processor " + StreamProcessorName + " deleted.");
    }
}
```

使用流处理器搜索人脸（Java V1 示例）

以下示例代码显示如何使用 Java V1 调用各种流处理器操作 StartStreamProcessor，例如CreateStreamProcessor和。该示例包括一个流处理器管理器类 (StreamManager)，该类提供调用流处理器操作的方法。入门类（Starter）创建一个 StreamManager 对象并调用各种操作。

配置示例:

1. 将 Starter 类成员字段的值设置为所需值。
2. 在 Starter 类函数 `main` 中，取消注释所需的函数调用。

Starter 类

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Starter class. Use to create a StreamManager class
// and call stream processor operations.
package com.amazonaws.samples;
import com.amazonaws.samples.*;

public class Starter {

 public static void main(String[] args) {
```

```
    String streamProcessorName="Stream Processor Name";
    String kinesisVideoStreamArn="Kinesis Video Stream Arn";
    String kinesisDataStreamArn="Kinesis Data Stream Arn";
    String roleArn="Role Arn";
    String collectionId="Collection ID";
    Float matchThreshold=50F;

  try {
   StreamManager sm= new StreamManager(streamProcessorName,
     kinesisVideoStreamArn,
     kinesisDataStreamArn,
     roleArn,
     collectionId,
     matchThreshold);
    //sm.createStreamProcessor();
    //sm.startStreamProcessor();
    //sm.deleteStreamProcessor();
    //sm.deleteStreamProcessor();
    //sm.stopStreamProcessor();
    //sm.listStreamProcessors();
    //sm.describeStreamProcessor();
  }
  catch(Exception e){
   System.out.println(e.getMessage());
  }
 }
}
```

StreamManager 班级

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Stream manager class. Provides methods for calling
// Stream Processor operations.
package com.amazonaws.samples;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateStreamProcessorRequest;
```

```
import com.amazonaws.services.rekognition.model.CreateStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorResult;
import com.amazonaws.services.rekognition.model.FaceSearchSettings;
import com.amazonaws.services.rekognition.model.KinesisDataStream;
import com.amazonaws.services.rekognition.model.KinesisVideoStream;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsRequest;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsResult;
import com.amazonaws.services.rekognition.model.StartStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StartStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StopStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StopStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StreamProcessor;
import com.amazonaws.services.rekognition.model.StreamProcessorInput;
import com.amazonaws.services.rekognition.model.StreamProcessorOutput;
import com.amazonaws.services.rekognition.model.StreamProcessorSettings;

public class StreamManager {

    private String streamProcessorName;
    private String kinesisVideoStreamArn;
    private String kinesisDataStreamArn;
    private String roleArn;
    private String collectionId;
    private float matchThreshold;

    private AmazonRekognition rekognitionClient;


    public StreamManager(String spName,
      String kvStreamArn,
      String kdStreamArn,
      String iamRoleArn,
      String collId,
      Float threshold){
     streamProcessorName=spName;
     kinesisVideoStreamArn=kvStreamArn;
     kinesisDataStreamArn=kdStreamArn;
     roleArn=iamRoleArn;
     collectionId=collId;
     matchThreshold=threshold;
     rekognitionClient=AmazonRekognitionClientBuilder.defaultClient();
```

```
   }

   public void createStreamProcessor() {
    //Setup input parameters
       KinesisVideoStream kinesisVideoStream = new
KinesisVideoStream().withArn(kinesisVideoStreamArn);
       StreamProcessorInput streamProcessorInput =
               new StreamProcessorInput().withKinesisVideoStream(kinesisVideoStream);
       KinesisDataStream kinesisDataStream = new
KinesisDataStream().withArn(kinesisDataStreamArn);
       StreamProcessorOutput streamProcessorOutput =
               new StreamProcessorOutput().withKinesisDataStream(kinesisDataStream);
       FaceSearchSettings faceSearchSettings =
               new
FaceSearchSettings().withCollectionId(collectionId).withFaceMatchThreshold(matchThreshold);
       StreamProcessorSettings streamProcessorSettings =
               new StreamProcessorSettings().withFaceSearch(faceSearchSettings);

       //Create the stream processor
       CreateStreamProcessorResult createStreamProcessorResult =
rekognitionClient.createStreamProcessor(
               new
CreateStreamProcessorRequest().withInput(streamProcessorInput).withOutput(streamProcessorOutpu

.withSettings(streamProcessorSettings).withRoleArn(roleArn).withName(streamProcessorName));

       //Display result
       System.out.println("Stream Processor " + streamProcessorName + " created.");
       System.out.println("StreamProcessorArn - " +
createStreamProcessorResult.getStreamProcessorArn());
   }

   public void startStreamProcessor() {
       StartStreamProcessorResult startStreamProcessorResult =
               rekognitionClient.startStreamProcessor(new
StartStreamProcessorRequest().withName(streamProcessorName));
       System.out.println("Stream Processor " + streamProcessorName + " started.");
   }

   public void stopStreamProcessor() {
       StopStreamProcessorResult stopStreamProcessorResult =
               rekognitionClient.stopStreamProcessor(new
StopStreamProcessorRequest().withName(streamProcessorName));
```

```
        System.out.println("Stream Processor " + streamProcessorName + " stopped.");
    }

    public void deleteStreamProcessor() {
        DeleteStreamProcessorResult deleteStreamProcessorResult = rekognitionClient
                .deleteStreamProcessor(new
DeleteStreamProcessorRequest().withName(streamProcessorName));
        System.out.println("Stream Processor " + streamProcessorName + " deleted.");
    }

    public void describeStreamProcessor() {
        DescribeStreamProcessorResult describeStreamProcessorResult = rekognitionClient
                .describeStreamProcessor(new
DescribeStreamProcessorRequest().withName(streamProcessorName));

        //Display various stream processor attributes.
        System.out.println("Arn - " +
describeStreamProcessorResult.getStreamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
                +
describeStreamProcessorResult.getInput().getKinesisVideoStream().getArn());
        System.out.println("Output kinesisData stream - "
                +
describeStreamProcessorResult.getOutput().getKinesisDataStream().getArn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.getRoleArn());
        System.out.println(
                "CollectionId - " +
describeStreamProcessorResult.getSettings().getFaceSearch().getCollectionId());
        System.out.println("Status - " + describeStreamProcessorResult.getStatus());
        System.out.println("Status message - " +
describeStreamProcessorResult.getStatusMessage());
        System.out.println("Creation timestamp - " +
describeStreamProcessorResult.getCreationTimestamp());
        System.out.println("Last update timestamp - " +
describeStreamProcessorResult.getLastUpdateTimestamp());
    }

    public void listStreamProcessors() {
        ListStreamProcessorsResult listStreamProcessorsResult =
                rekognitionClient.listStreamProcessors(new
ListStreamProcessorsRequest().withMaxResults(100));

        //List all stream processors (and state) returned from Rekognition
```

```
        for (StreamProcessor streamProcessor :
 listStreamProcessorsResult.getStreamProcessors()) {
            System.out.println("StreamProcessor name - " + streamProcessor.getName());
            System.out.println("Status - " + streamProcessor.getStatus());
        }
    }
}
```

读取流视频分析结果

您可使用 Amazon Kinesis Data Streams 客户端库来使用已发送到 Amazon Kinesis Data Streams 输出流的分析结果。有关更多信息，请参阅从 Kinesis 数据流读取数据。Amazon Rekognition Video 将每个分析过的帧的 JSON 帧记录放入 Kinesis 输出流。Amazon Rekognition Video 不会分析通过 Kinesis 视频流传递给它的每一帧。

已发送到 Kinesis 数据流的帧记录包含以下信息：该帧位于哪个 Kinesis 视频流片段中、该帧位于片段中的哪个位置以及在该帧中识别的人脸。它还包含流处理器的状态信息。有关更多信息，请参阅 了解 Kinesis 人脸识别 JSON 帧记录。

Amazon Kinesis Video Streams 解析器库包含使用 Amazon Rekognition Video 结果并将其与原始 Kinesis 视频流集成的示例测试。有关更多信息，请参阅 在本地使用 Kinesis 视频流显示 Rekognition 结果。

Amazon Rekognition Video 将 Amazon Rekognition Video 的分析信息流式传输到 Kinesis 数据流。以下是单个记录的 JSON 示例。

```
{
  "InputInformation": {
    "KinesisVideo": {
      "StreamArn": "arn:aws:kinesisvideo:us-west-2:nnnnnnnnnnnn:stream/stream-name",
      "FragmentNumber": "91343852333289682796718532614445757584843717598",
      "ServerTimestamp": 1510552593.455,
      "ProducerTimestamp": 1510552593.193,
      "FrameOffsetInSeconds": 2
    }
  },
  "StreamProcessorInformation": {
    "Status": "RUNNING"
  },
  "FaceSearchResponse": [
    {
      "DetectedFace": {
```

```
        "BoundingBox": {
          "Height": 0.075,
          "Width": 0.05625,
          "Left": 0.428125,
          "Top": 0.40833333
        },
        "Confidence": 99.975174,
        "Landmarks": [
          {
            "X": 0.4452057,
            "Y": 0.4395594,
            "Type": "eyeLeft"
          },
          {
            "X": 0.46340984,
            "Y": 0.43744427,
            "Type": "eyeRight"
          },
          {
            "X": 0.45960626,
            "Y": 0.4526856,
            "Type": "nose"
          },
          {
            "X": 0.44958648,
            "Y": 0.4696949,
            "Type": "mouthLeft"
          },
          {
            "X": 0.46409217,
            "Y": 0.46704912,
            "Type": "mouthRight"
          }
        ],
        "Pose": {
          "Pitch": 2.9691637,
          "Roll": -6.8904796,
          "Yaw": 23.84388
        },
        "Quality": {
          "Brightness": 40.592964,
          "Sharpness": 96.09616
        }
      },
```

```
      "MatchedFaces": [
        {
          "Similarity": 88.863960,
          "Face": {
            "BoundingBox": {
              "Height": 0.557692,
              "Width": 0.749838,
              "Left": 0.103426,
              "Top": 0.206731
            },
            "FaceId": "ed1b560f-d6af-5158-989a-ff586c931545",
            "Confidence": 99.999201,
            "ImageId": "70e09693-2114-57e1-807c-50b6d61fa4dc",
            "ExternalImageId": "matchedImage.jpeg"
          }
        }
      ]
    }
  ]
}
```

在 JSON 示例中，请注意以下内容：

- InputInformation— 有关用于将视频流式传输到亚马逊 Rekognition Video 的 Kinesis 视频流的信息。有关更多信息，请参阅 InputInformation。
- StreamProcessorInformation— 亚马逊 Rekognition Video 流处理器的状态信息。`Status` 字段的唯一可能值为 RUNNING。有关更多信息，请参阅 StreamProcessorInformation。
- FaceSearchResponse— 包含有关流视频中与输入集合中的人脸匹配的人脸的信息。FaceSearchResponse包含一个DetectedFace对象，即在分析的视频帧中检测到的人脸。对于每个检测到的人脸，数组 `MatchedFaces` 包含一组在输入集合中找到的匹配的人脸对象 (MatchedFace) 以及一个相似度得分。

将 Kinesis 视频流映射到 Kinesis 数据流

您可能希望将 Kinesis 视频流帧映射到发送至 Kinesis 数据流的分析帧。例如，在流视频显示期间，您可能希望在识别出的人脸周围显示方框。边界框的坐标将作为 Kinesis 人脸识别记录的一部分发送给 Kinesis 数据流。为了正确显示边界框，您需要将随 Kinesis 人脸识别记录发送的时间信息映射到源 Kinesis 视频流中相应的帧。

将 Kinesis 视频流映射到 Kinesis 数据流所用的方法取决于流式处理的是实时媒体（如实时流视频）还是存档媒体（如存储的视频）。

流式处理实时媒体时的映射

将 Kinesis 视频流帧映射到 Kinesis 数据流帧中

1. 将PutMedia操作FragmentTimeCodeType的输入参数设置为RELATIVE。
2. 调用 PutMedia 将实时媒体传输到 Kinesis 视频流中。
3. 如果从 Kinesis 数据流接收 Kinesis 人脸识别记录，请存储 KinesisVideo 字段的 ProducerTimestamp 和 FrameOffsetInSeconds 值。
4. 同时添加 ProducerTimestamp 和 FrameOffsetInSeconds 字段值，计算与 Kinesis 视频流帧对应的时间戳。

流式处理存档媒体时的映射

将 Kinesis 视频流帧映射到 Kinesis 数据流帧中

1. 致电PutMedia将存档的媒体传输到 Kinesis 视频流中。
2. 如果您从 PutMedia 操作的响应中接收到 Acknowledgement 对象，请存储 Payload 字段的 FragmentNumber 字段值。FragmentNumber 是 MKV 集群的片段编号。
3. 如果从 Kinesis 数据流接收 Kinesis 人脸识别记录，请存储 KinesisVideo 字段的 FrameOffsetInSeconds 字段值。
4. 使用步骤 2 和步骤 3 中存储的 FrameOffsetInSeconds 和 FragmentNumber 值计算映射。FrameOffsetInSeconds 是发送至 Amazon Kinesis Data Streams 的具有特定 FragmentNumber 的片段中的偏移。有关获取已知片段编号的视频帧的更多信息，请参阅 Amazon Kinesis Video Streams 存档媒体。

在本地使用 Kinesis 视频流显示 Rekognition 结果

你可以使用亚马逊 Kinesis Video Streams 解析器库的示例测试，查看亚马逊 Rekognition Video Streams 的结果显示在你来自亚马逊 Kinesis Video Streams 的提要中。KinesisVideo 在检测到的人脸上方KinesisVideoRekognitionIntegrationExample显示边界框，并通过本地渲染视频。JFrame此过程假设您已成功将来自设备摄像头的媒体输入连接到 Kinesis 视频流并启动了 Amazon Rekognition 流处理器。有关更多信息，请参阅 使用 GStreamer插件进行直播。

步骤 1：安装 Kinesis 视频流解析器库

要创建目录并下载 Github 存储库，请运行以下命令：

```
$ git clone https://github.com/aws/amazon-kinesis-video-streams-parser-library.git
```

导航到库目录并运行以下 Maven 命令进行全新安装：

```
$ mvn clean install
```

步骤 2：配置 Kinesis 视频流和 Rekognition 集成示例测试

打开 KinesisVideoRekognitionIntegrationExampleTest.java文件。删除类标题后的 @Ignore。使用来自 Amazon Kinesis 和 Amazon Rekognition 资源的信息填充数据字段。有关更多信息，请参阅 设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源。如果您要将视频流式传输到 Kinesis 视频流，请移除 inputStream 参数。

请看下面的代码示例:

```
RekognitionInput rekognitionInput = RekognitionInput.builder()
  .kinesisVideoStreamArn("arn:aws:kinesisvideo:us-east-1:123456789012:stream/
rekognition-test-video-stream")
  .kinesisDataStreamArn("arn:aws:kinesis:us-east-1:123456789012:stream/
AmazonRekognition-rekognition-test-data-stream")
  .streamingProcessorName("rekognition-test-stream-processor")
  // Refer how to add face collection :
  // https://docs.aws.amazon.com/rekognition/latest/dg/add-faces-to-collection-
procedure.html
  .faceCollectionId("rekognition-test-face-collection")
  .iamRoleArn("rekognition-test-IAM-role")
  .matchThreshold(0.95f)
  .build();

KinesisVideoRekognitionIntegrationExample example =
 KinesisVideoRekognitionIntegrationExample.builder()
  .region(Regions.US_EAST_1)
  .kvsStreamName("rekognition-test-video-stream")
  .kdsStreamName("AmazonRekognition-rekognition-test-data-stream")
  .rekognitionInput(rekognitionInput)
  .credentialsProvider(new ProfileCredentialsProvider())
```

```
 // NOTE: Comment out or delete the inputStream parameter if you are streaming video,
otherwise
 // the test will use a sample video.
 //.inputStream(TestResourceUtil.getTestInputStream("bezos_vogels.mkv"))
 .build();
```

步骤 3：运行 Kinesis 视频流和 Rekognition 集成示例测试

如果您正在向其进行流式传输，请确保您的 Kinesis 视频流正在接收媒体输入，然后在运行 Amazon Rekognition Video 流处理器的情况下开始分析您的流。有关更多信息，请参阅 Amazon Rekognition Video 流处理器操作概述。将`KinesisVideoRekognitionIntegrationExampleTest`课程当作 JUnit测试来运行。短暂延迟后，将打开一个新窗口，其中包含来自您的 Kinesis 视频流的视频源，以及在检测到的人脸上绘制的边界框。

> **Note**
>
> 本示例中使用的集合中的人脸必须以这种格式指定外部图像 ID（文件名），边界框标签才能显示有意义的文本： PersonName1-Trusted、 PersonName 2-Intruder、 PersonName 3-Neutral 等。标签也可以采用颜色编码，并且可以在 FaceType .java 文件中进行自定义。

了解 Kinesis 人脸识别 JSON 帧记录

您可以使用 Amazon Rekognition Video 来识别流视频中的人脸。对于每个分析过的帧，Amazon Rekognition Video 会将 JSON 帧记录输出到 Kinesis 数据流中。Amazon Rekognition Video 不会分析通过 Kinesis 视频流传递给它的每一帧。

JSON 帧记录包含以下信息：输入和输出流、流处理器的状态以及在分析过的帧中识别的人脸。本节包含 JSON 帧记录的参考信息。

以下是 Kinesis 数据流记录的 JSON 语法。有关更多信息，请参阅 使用流视频事件。

> **Note**
>
> Amazon Rekognition Video API 的工作原理是将输入流中的人脸与人脸集合进行比较，并返回找到的最接近的匹配项以及相似度分数。

```
{
```

```
    "InputInformation": {
        "KinesisVideo": {
            "StreamArn": "string",
            "FragmentNumber": "string",
            "ProducerTimestamp": number,
            "ServerTimestamp": number,
            "FrameOffsetInSeconds": number
        }
    },
    "StreamProcessorInformation": {
        "Status": "RUNNING"
    },
    "FaceSearchResponse": [
        {
            "DetectedFace": {
                "BoundingBox": {
                    "Width": number,
                    "Top": number,
                    "Height": number,
                    "Left": number
                },
                "Confidence": number,
                "Landmarks": [
                    {
                        "Type": "string",
                        "X": number,
                        "Y": number
                    }
                ],
                "Pose": {
                    "Pitch": number,
                    "Roll": number,
                    "Yaw": number
                },
                "Quality": {
                    "Brightness": number,
                    "Sharpness": number
                }
            },
            "MatchedFaces": [
                {
                    "Similarity": number,
                    "Face": {
                        "BoundingBox": {
```

```
                            "Width": number,
                            "Top": number,
                            "Height": number,
                            "Left": number
                        },
                        "Confidence": number,
                        "ExternalImageId": "string",
                        "FaceId": "string",
                        "ImageId": "string"
                    }
                }
            ]
        }
    ]
}
```

JSON 记录

JSON 记录包含有关由 Amazon Rekognition Video 处理的帧的信息。该记录包含有关流视频的信息、分析过的帧的状态信息以及有关在该帧中识别的人脸的信息。

InputInformation

有关用于将视频流式传输到 Amazon Rekognition Video 的 Kinesis 视频流的信息。

类型：InputInformation 对象

StreamProcessorInformation

有关 Amazon Rekognition Video 流处理器的信息。这包括流处理器的当前状态的状态信息。

类型：StreamProcessorInformation 对象

FaceSearchResponse

有关在流视频帧中检测到的人脸与在输入集合中找到的匹配人脸的信息。

类型：FaceSearchResponse 对象数组

InputInformation

有关 Amazon Rekognition Video 使用的源视频流的信息。有关更多信息，请参阅 使用流视频事件。

KinesisVideo

类型：KinesisVideo 对象

KinesisVideo

有关将源视频流式传输到 Amazon Rekognition Video 的 Kinesis 视频流的信息。有关更多信息，请参阅 使用流视频事件。

StreamArn

Kinesis 数据流的 Amazon 资源名称 (ARN)。

类型：字符串

FragmentNumber

一个流视频的片断，包含此记录表示的帧。

类型：字符串

ProducerTimestamp

片段的生成者端 Unix 时间戳。有关更多信息，请参阅 PutMedia。

类型：数字

ServerTimestamp

片段的服务器端 Unix 时间戳。有关更多信息，请参阅 PutMedia。

类型：数字

FrameOffsetInSeconds

片段内的帧的偏移量（以秒为单位）。

类型：数字

StreamProcessorInformation

有关流处理器的状态信息。

状态

流处理器的当前状态。一个可能的值为 RUNNING。

类型：字符串

FaceSearchResponse

有关在流视频帧中检测到的人脸与集合中的与检测到的人脸匹配的人脸的信息。您可在调用 CreateStreamProcessor 时指定集合。有关更多信息，请参阅 使用流视频事件。

DetectedFace

在分析的视频帧中检测到的人脸的人脸详细信息。

类型：DetectedFace 对象

MatchedFaces

集合中的人脸的人脸详细信息数组，该集合与在 DetectedFace 中检测到的人脸匹配。

类型：MatchedFace 对象数组

DetectedFace

有关在流视频帧中检测到的人脸的信息。MatchedFace 对象字段中提供了输入集合中的匹配人脸。

BoundingBox

在分析过的视频帧内检测到的人脸的边界框坐标。该 BoundingBox 对象与用于图像分析的 BoundingBox 对象具有相同的属性。

类型：BoundingBox 对象

置信度

Amazon Rekognition Video 对检测到的人脸是否真的是人脸的置信度 (1-100)。1 表示最低置信度，100 表示最高置信度。

类型：数字

标记

一组人脸标记。

类型：地标对象数组

姿势

指示根据人脸的俯仰、翻滚和偏转确定的人脸的姿势。

类型：姿势对象

质量

确定人脸图像亮度和锐度。

类型：ImageQuality 对象

MatchedFace

有关与在分析的视频帧中检测到的人脸匹配的人脸的信息。

人脸

人脸匹配信息，针对输入集合中与 DetectedFace 对象中的人脸匹配的人脸。

类型：人脸对象

相似度

人脸匹配的置信度 (1-100)。1 表示最低置信度，100 表示最高置信度。

类型：数字

使用 GStreamer插件进行直播

Amazon Rekognition Video 可以分析来自设备摄像头的实时流视频。要访问来自设备来源的媒体输入，你需要安装 GStreamer。 GStreamer 是一款第三方多媒体框架软件，可在工作流管道中将媒体源和处理工具连接在一起。您还需要安装适用于 Gstreamer 的 Amazon Kinesis Video Streams Producer 插件。此过程假设您已成功设置 Amazon Rekognition Video 和 Amazon Kinesis 资源。有关更多信息，请参阅 设置您的 Amazon Rekognition Video 和 Amazon Kinesis 资源。

步骤 1：安装 Gstreamer

下载并安装第三方多媒体平台软件 Gstreamer。您可以使用 Homebrew (Homebrew 上的 Gstreamer) 等软件包管理软件，或直接从 Freedesktop 网站获取。

通过从命令行终端启动带有测试源的视频源，验证 Gstreamer 的安装是否成功。

```
$ gst-launch-1.0 videotestsrc ! autovideosink
```

步骤 2：安装 Kinesis Video Streams Producer 插件

在本节中，您将下载 Amazon Kinesis Video Streams Producer Library 并安装 Kinesis Video Streams Gstreamer 插件。

创建一个目录，然后从 GitHub 存储库克隆源代码。请务必包含 `--recursive` 参数。

```
$ git clone --recursive https://github.com/awslabs/amazon-kinesis-video-streams-
producer-sdk-cpp.git
```

按照库提供的说明配置和构建项目。请务必使用适用于操作系统的平台专用命令。运行 `cmake` 安装 Kinesis Video Streams Gstreamer 插件时，请使用 `-DBUILD_GSTREAMER_PLUGIN=ON` 参数。该项目需要安装中包含的以下其他软件包：GCC 或 Clang、Curl、Openssl 和 Log4cplus。如果由于缺少软件包而导致构建失败，请验证该软件包是否已安装并在您的 PATH 中。如果您在构建时遇到“无法运行 C 编译的程序”错误，请再次运行构建命令。有时，无法找到正确的 C 编译器。

运行以下命令验证 Kinesis Video Streams 插件的安装。

```
$ gst-inspect-1.0 kvssink
```

应显示以下信息，例如出厂和插件详细信息：

```
Factory Details:
  Rank                     primary + 10 (266)
  Long-name                KVS Sink
  Klass                    Sink/Video/Network
  Description              GStreamer AWS KVS plugin
  Author                   AWS KVS <kinesis-video-support@amazon.com>

Plugin Details:
  Name                     kvssink
  Description              GStreamer AWS KVS plugin
  Filename                 /Users/YOUR_USER/amazon-kinesis-video-streams-producer-sdk-
cpp/build/libgstkvssink.so
  Version                  1.0
  License                  Proprietary
  Source module            kvssinkpackage
  Binary package           GStreamer
```

```
  Origin URL              http://gstreamer.net/

  ...
```

步骤 3：使用 Kinesis Video Streams 插件运行 Gstreamer

在开始从设备摄像头流式传输到 Kinesis 视频流之前，可能需要将媒体源转换为 Kinesis 视频流可接受的编解码器。要确定当前已连接到您的计算机的设备的规格和格式功能，请运行以下命令。

```
$ gst-device-monitor-1.0
```

要开始流式传输，请使用以下示例命令启动 Gstreamer，并添加您的凭证和 Amazon Kinesis Video Streams 信息。在授予 Amazon Rekognition 访问您的 Kinesis 流的权限时，您应该使用您创建的 IAM 服务角色的访问密钥和区域。有关访问密钥的更多信息，请参阅《IAM 用户指南》中的管理 IAM 用户的访问密钥。此外，您还可以根据使用要求调整视频格式参数，这些参数可从您的设备中获取。

```
$ gst-launch-1.0 autovideosrc device=/dev/video0 ! videoconvert ! video/x-
raw,format=I420,width=640,height=480,framerate=30/1 !
                x264enc bframes=0 key-int-max=45 bitrate=500 ! video/x-h264,stream-
format=avc,alignment=au,profile=baseline !
                kvssink stream-name="YOUR_STREAM_NAME" storage-size=512 access-
key="YOUR_ACCESS_KEY" secret-key="YOUR_SECRET_ACCESS_KEY" aws-region="YOUR_AWS_REGION"
```

有关更多启动命令，请参阅 GStreamer 启动命令示例。

> **Note**
>
> 如果您的启动命令因非协商错误而终止，请检查设备显示器的输出，并确保 `videoconvert` 参数值是设备的有效功能。

几秒钟后，您将在 Kinesis 视频流中看到来自设备摄像头的视频源。要开始使用 Amazon Rekognition 检测和匹配人脸，请启动您的 Amazon Rekognition Video 流处理器。有关更多信息，请参阅 Amazon Rekognition Video 流处理器操作概述。

流视频问题排查

本主题提供有关将 Amazon Rekognition Video 与流视频结合使用的问题排查信息。

主题

- 我不知道我的流处理器是否已成功创建
- 我不知道我是否已正确配置我的流处理器
- 我的流处理器未返回结果
- 我的流处理器的状态为 FAILED
- 我的流处理器未返回预期结果

我不知道我的流处理器是否已成功创建

使用以下 AWS CLI 命令获取流处理器及其当前状态的列表。

```
aws rekognition list-stream-processors
```

您可以使用以下 AWS CLI 命令获取更多详细信息。将 `stream-processor-name` 替换为所需流处理器的名称。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

我不知道我是否已正确配置我的流处理器

如果您的代码未输出来自 Amazon Rekognition Video 的分析结果，则您的流处理器可能未正确配置。请执行以下操作来确认您的流处理器已正确配置并能够生成结果。

确定您的解决方案是否已正确配置

1. 运行以下命令来确认您的流处理器处于正在运行状态。将 `stream-processor-name` 更改为您的流处理器的名称。如果 `Status` 的值为 `RUNNING`，则表示流处理器正在运行。如果状态为 `RUNNING` 并且您未获得结果，请参阅 我的流处理器未返回结果。如果状态为 `FAILED`，请参阅我的流处理器的状态为 FAILED。

   ```
   aws rekognition describe-stream-processor --name stream-processor-name
   ```

2. 如果您的流处理器正在运行，请运行以下 Bash 或 PowerShell 命令从输出 Kinesis 数据流中读取数据。

   Bash

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000
 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-data-stream-name --query
 'ShardIterator')
                        aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

PowerShell

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-
id shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-
data-stream-name).split('"')[4])
```

3. 使用 Base64 解码网站上的解码工具将输出解码为人类可读的字符串。有关更多信息，请参阅步骤 3：获取记录。
4. 如果命令有效，并且您在 Kinesis 数据流看到人脸检测结果，则表示您的解决方案已正确配置。如果命令失败，请查看其他问题排查建议并参阅允许 Amazon Rekognition Video 访问您的资源。

或者，您可以使用 “kinesis-process-record” AWS Lambda 蓝图将来自 Kinesis 数据流的消息记录到以 CloudWatch 实现持续可视化。这会产生 AWS Lambda 和 CloudWatch的额外成本。

我的流处理器未返回结果

出于以下几个原因，您的流处理器可能不会返回结果。

原因 1：您的流处理器未正确配置

您的流处理器可能未进行正确配置。有关更多信息，请参阅 我不知道我是否已正确配置我的流处理器。

原因 2：您的流处理器未处于 RUNNING 状态

排查流处理器状态的问题

1. 使用以下 AWS CLI 命令检查流处理器的状态。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. 如果 Status 的值为 STOPPED，则使用以下命令启动您的流处理器：

```
aws rekognition start-stream-processor --name stream-processor-name
```

3. 如果 Status 的值为 FAILED，请参阅我的流处理器的状态为 FAILED。
4. 如果 Status 的值为 STARTING，则等待 2 分钟，然后重复步骤 1 以检查状态。如果 Status 的值仍为 STARTING，请执行以下操作：

   a. 使用以下命令删除流处理器。

   ```
   aws rekognition delete-stream-processor --name stream-processor-name
   ```

   b. 使用相同配置创建新的流处理器。有关更多信息，请参阅 使用流视频事件。

   c. 如果您仍然遇到问题，请联系 Su AWS pport。
5. 如果 Status 的值为 RUNNING，请参阅原因 3：Kinesis 视频流中没有有效数据。

原因 3：Kinesis 视频流中没有有效数据

检查 Kinesis 视频流中是否有有效数据

1. 登录并打开 Amazon Kinesis Video Streams 控制台 https://console.aws.amazon.com/kinesisvideo/，网址为。 AWS Management Console
2. 选择作为 Amazon Rekognition 流处理器输入的 Kinesis 视频流。
3. 如果预览表明流上无数据，则 Amazon Rekognition Video 的输入流中没有要处理的任何数据。

有关使用 Kinesis 视频流生成视频的信息，请参阅 Kinesis 视频流创建者库。

我的流处理器的状态为 FAILED

您可以使用以下 AWS CLI 命令检查流处理器的状态。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

如果 Status 的值为 FAILED，则查看以下错误消息的问题排查信息。

错误：“（对角色的访问被拒绝）”

流处理器所用的 IAM 角色不存在或 Amazon Rekognition Video 无权代入该角色。

排查对 IAM 角色的访问权限问题

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。

2. 在左侧导航窗格中，选择角色并确认该角色存在。
3. 如果该角色存在，请检查该角色是否具有AmazonRekognitionServiceRole权限策略。
4. 如果该角色不存在或没有相应的权限，请参阅 允许 Amazon Rekognition Video 访问您的资源。
5. 使用以下 AWS CLI 命令启动流处理器。

```
aws rekognition start-stream-processor --name stream-processor-name
```

错误：“对 Kinesis 视频的访问被拒绝或 对 Kinesis 数据的访问被拒绝”

该角色无权访问 Kinesis Video Streams API 操作 `GetMedia` 和 `GetDataEndpoint`。此外，它可能无权访问 Kinesis Data Streams API 操作 `PutRecord` 和 `PutRecords`。

排查 API 权限问题

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 打开该角色并确保它已附加以下权限策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "data-arn"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "video-arn"
        }
    ]
}
```

3. 如果缺少任何权限，请更新该策略。有关更多信息，请参阅 允许 Amazon Rekognition Video 访问您的资源。

错误：“直播*input-video-stream-name*不存在”

针对流处理器的 Kinesis 视频流输入不存在或未正确配置。

排查 Kinesis 视频流问题

1. 使用以下命令确认该流存在。

```
aws kinesisvideo list-streams
```

2. 如果该流存在，请检查以下各项。

   - Amazon 资源名称 (ARN) 与流处理器的输入流的 ARN 相同。
   - Kinesis 视频流必须位于与流处理器相同的区域中。

   如果流处理器配置不正确，请使用以下 AWS CLI 命令将其删除。

```
aws rekognition delete-stream-processor --name stream-processor-name
```

3. 使用预期的 Kinesis 视频流创建新的流处理器。有关更多信息，请参阅 创建 Amazon Rekognition Video 人脸搜索流处理器。

错误：“找不到集合”

流处理器用于人脸匹配的 Amazon Rekognition 集合不存在或使用的是错误集合。

确认集合

1. 使用以下 AWS CLI 命令来确定所需的集合是否存在。切换`region`到你运行流处理器的 AWS 区域。

```
aws rekognition list-collections --region region
```

   如果所需的集合不存在，则创建新的集合并添加人脸信息。有关更多信息，请参阅 在集合中搜索人脸。

2. 在您调用 CreateStreamProcessor 时，请检查 `CollectionId` 输入参数的值是否正确。

3. 使用以下 AWS CLI 命令启动流处理器。

```
aws rekognition start-stream-processor --name stream-processor-name
```

错误：“找*account-id*不到账号*output-kinesis-data-stream-name*下的直播”

流处理器使用的输出 Kinesis 数据流在您的区域中不存在 AWS 账户 或与您的流处理器不在同一个 AWS 区域。

排查 Kinesis 数据流问题

1. 使用以下 AWS CLI 命令确定 Kinesis 数据流是否存在。切换`region`到您使用流媒体处理器的 AWS 区域。

```
aws kinesis list-streams --region region
```

2. 如果 Kinesis 数据流存在，请检查 Kinesis 数据流名称是否与流处理器所用的输出流的名称相同。
3. 如果 Kinesis 数据流不存在，则它可能存在于其他 AWS 区域。Kinesis 数据流必须位于与流处理器相同的区域中。
4. 如有必要，请创建一个新的 Kinesis 数据流。
   a. 使用与流处理器所用的名称相同的名称创建 Kinesis 数据流。有关更多信息，请参阅步骤 1：创建数据流。
   b. 使用以下 AWS CLI 命令启动流处理器。

```
aws rekognition start-stream-processor --name stream-processor-name
```

我的流处理器未返回预期结果

如果您的流处理器未返回预期的人脸匹配，请使用以下信息。

- 在集合中搜索人脸
- 针对摄像机设置（流视频）的建议

人物的轨迹

> **Note**
>
> 终止支持通知：2025 年 10 月 31 日，AWS 将停止对 Amazon Rekognition 人物轨迹跟踪的支持。2025 年 10 月 31 日之后，您将无法再使用 Rekognition 人物轨迹跟踪功能。有关更多信息，请访问此博客文章。

Amazon Rekognition Video 可以创建视频中的人物轨迹并提供如下信息：

- 在跟踪人物的轨迹时人物在视频帧中的位置。
- 人脸标记，如左眼的位置（如果检测到）。

存储视频中的 Amazon Rekognition Video 人物轨迹跟踪是一个异步操作。要开始在视频中寻找人物路径，请致电StartPersonTracking。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service 主题。如果视频分析成功，请致电GetPersonTracking以获取视频分析的结果。有关调用 Amazon Rekognition Video API 操作的更多信息，请参阅 调用 Amazon Rekognition Video 操作。

以下过程说明如何通过存储在 Amazon S3 存储桶内的视频跟踪人物轨迹。此示例扩展了 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon Simple Queue Service 队列获取视频分析请求的完成状态）中的代码。

检测存储在 Amazon S3 存储桶内的视频中的人员 (SDK)

1. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
2. 将以下代码添加到您在步骤 1 中创建的类 `VideoDetect`。

   Java

```
      //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights
 Reserved.
      //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

       //
Persons======================================================================
```

```
       private static void StartPersonDetection(String bucket, String video)
throws Exception{


           NotificationChannel channel= new NotificationChannel()
                   .withSNSTopicArn(snsTopicArn)
                   .withRoleArn(roleArn);

        StartPersonTrackingRequest req = new StartPersonTrackingRequest()
                .withVideo(new Video()
                        .withS3Object(new S3Object()
                            .withBucket(bucket)
                            .withName(video)))
                .withNotificationChannel(channel);



        StartPersonTrackingResult startPersonDetectionResult =
rek.startPersonTracking(req);
        startJobId=startPersonDetectionResult.getJobId();

       }

       private static void GetPersonDetectionResults() throws Exception{
           int maxResults=10;
           String paginationToken=null;
           GetPersonTrackingResult personTrackingResult=null;

           do{
               if (personTrackingResult !=null){
                   paginationToken = personTrackingResult.getNextToken();
               }

               personTrackingResult = rek.getPersonTracking(new
GetPersonTrackingRequest()
                    .withJobId(startJobId)
                    .withNextToken(paginationToken)
                    .withSortBy(PersonTrackingSortBy.TIMESTAMP)
                    .withMaxResults(maxResults));

               VideoMetadata
videoMetaData=personTrackingResult.getVideoMetadata();

               System.out.println("Format: " + videoMetaData.getFormat());
```

```
                System.out.println("Codec: " + videoMetaData.getCodec());
                System.out.println("Duration: " +
videoMetaData.getDurationMillis());
                System.out.println("FrameRate: " +
videoMetaData.getFrameRate());


                //Show persons, confidence and detection times
                List<PersonDetection> detectedPersons=
personTrackingResult.getPersons();

                for (PersonDetection detectedPerson: detectedPersons) {

                   long seconds=detectedPerson.getTimestamp()/1000;
                   System.out.print("Sec: " + Long.toString(seconds) + " ");
                   System.out.println("Person Identifier: "  +
detectedPerson.getPerson().getIndex());
                      System.out.println();
                }
            }  while (personTrackingResult !=null &&
personTrackingResult.getNextToken() != null);

        }
```

在函数 `main` 中，将以下行:

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

替换为:

```
        StartPersonDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetPersonDetectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoPersonDetection {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
```

```
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startPersonLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();
```

```
            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                    .jobTag("DetectingLabels")
                    .video(vidOb)
                    .notificationChannel(channel)
                    .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getPersonDetectionResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetPersonTrackingResponse personTrackingResult = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (personTrackingResult != null)
                    paginationToken = personTrackingResult.nextToken();

                GetPersonTrackingRequest recognitionRequest =
GetPersonTrackingRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds
                while (!finished) {

                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
```

```
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 personTrackingResult.videoMetadata();

                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<PersonDetection> detectedPersons =
 personTrackingResult.persons();
                for (PersonDetection detectedPerson : detectedPersons) {
                    long seconds = detectedPerson.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
 detectedPerson.person().index());
                    System.out.println();
                }

            } while (personTrackingResult != null &&
 personTrackingResult.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== People pathing ===============
    def StartPersonPathing(self):
        response=self.rek.start_person_tracking(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetPersonPathingResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_person_tracking(JobId=self.startJobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for personDetection in response['Persons']:
                print('Index: ' + str(personDetection['Person']['Index']))
                print('Timestamp: ' + str(personDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函数 main 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    analyzer.StartPersonPathing()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetPersonPathingResults()
```

CLI

运行以下 AWS CLI 命令开始人们在视频中寻找路径。

```
aws rekognition start-person-tracking --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-ARN","RoleArn":"role-ARN"}" \
--region region-name --profile profile-name
```

更新以下值：

- 将amzn-s3-demo-bucket和video-name更改为您在步骤 2 中指定的 Amazon S3 存储桶名称和文件名。
- 将 region-name 更改为您使用的 AWS 区域。
- 将创建 Rekognition 会话的行中的profile-name值替换为您的开发人员资料的名称。
- 将 topic-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 3 中创建的 Amazon SNS 主题的 ARN。
- 将 role-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 7 中创建的 IAM 服务角色的 ARN。

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。有关示例，请参阅以下内容：

```
aws rekognition start-person-tracking --video "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"video-name\"}}"
```

```
--notification-channel "{\"SNSTopicArn\":\"topic-ARN\",\"RoleArn\":\"role-ARN
\"}" \
--region region-name --profile profile-name
```

运行后续代码示例后，复制返回的 jobID 并将其提供给以下 GetPersonTracking 命令以获取结果，将 job-id-number 替换为您之前收到的 jobID：

```
aws rekognition get-person-tracking --job-id job-id-number
```

> **Note**
>
> 如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

3. 运行该代码。将显示被跟踪人员的唯一标识符以及跟踪人物轨迹的时间（秒）。

GetPersonTracking 操作响应

GetPersonTracking 将返回一个数组 Persons，其中包括 PersonDetection 对象，这些对象包含有关所检测人物的详细信息以及跟踪人物轨迹的时间。

您可通过使用 SortBy 输入参数来为 Persons 排序。指定 TIMESTAMP 以按在视频中检测人物轨迹的时间为元素排序。指定 INDEX 以按在视频中跟踪的人员排序。在每组人物的结果中，元素是按人物轨迹跟踪的准确性的置信度的降序顺序排序的。默认情况下，Persons 将按 TIMESTAMP 返回/排序。以下示例是 GetPersonDetection 的 JSON 响应。结果按在视频中跟踪人物轨迹的时间（播放视频的毫秒数）排序。在响应中，请注意以下内容：

- 人员信息 – PersonDetection 数组元素包含有关检测到的人员的信息。例如，检测到人员的时间（Timestamp）、检测到时人员在视频帧中的位置（BoundingBox）以及 Amazon Rekognition Video 对正确检测到人员的置信度（Confidence）。

  跟踪人物轨迹的每个时间戳不会返回面部特征。而且，在被跟踪人员的身体可能不可见的某些情况下，仅返回其脸部位置。

- 分页信息 – 此示例显示一页的人员检测信息。您可以在 GetPersonTracking 的 MaxResults 输入参数中指定要返回的人员元素数量。如果存在的结果的数量超过了 MaxResults，则 GetPersonTracking 会返回一个令牌 (NextToken)，用于获取下一页的结果。有关更多信息，请参阅 获取 Amazon Rekognition Video 分析结果。
- 索引 – 用于在整个视频中识别人物的唯一标识符。
- 视频信息 – 此响应包含有关由 GetPersonDetection 返回的每页信息中的视频格式（VideoMetadata）的信息。


```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "AcDymG0fSSoaI6+BBYpka5wVlqttysSPP8VvWcujMDluj1QpFo/vf
+mrMoqBGk8eUEiFlllR6g==",
    "Persons": [
        {
            "Person": {
                "BoundingBox": {
                    "Height": 0.8787037134170532,
                    "Left": 0.00572916679084301,
                    "Top": 0.12129629403352737,
                    "Width": 0.21666666865348816
                },
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20000000298023224,
                        "Left": 0.029999999329447746,
                        "Top": 0.2199999988079071,
                        "Width": 0.11249999701976776
                    },
                    "Confidence": 99.85971069335938,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.06842322647571564,
                            "Y": 0.3010137975215912
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.10543643683195114,
                            "Y": 0.29697132110595703
                        },
                        {
```

```
                        "Type": "nose",
                        "X": 0.09569807350635529,
                        "Y": 0.33701086044311523
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.0732642263174057,
                        "Y": 0.3757539987564087
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.10589495301246643,
                        "Y": 0.3722417950630188
                    }
                ],
                "Pose": {
                    "Pitch": -0.5589138865470886,
                    "Roll": -5.1093974113464355,
                    "Yaw": 18.69594955444336
                },
                "Quality": {
                    "Brightness": 43.052337646484375,
                    "Sharpness": 99.68138885498047
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.9074074029922485,
                "Left": 0.24791666865348816,
                "Top": 0.09259258955717087,
                "Width": 0.375
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "Top": 0.16333332657814026,
                    "Width": 0.12937499582767487
                },
```

```
                    "Confidence": 99.97504425048828,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.46415066719055176,
                            "Y": 0.2572723925113678
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.5068183541297913,
                            "Y": 0.23705792427062988
                        },
                        {
                            "Type": "nose",
                            "X": 0.49765899777412415,
                            "Y": 0.28383663296699524
                        },
                        {
                            "Type": "mouthLeft",
                            "X": 0.487221896648407,
                            "Y": 0.3452930748462677
                        },
                        {
                            "Type": "mouthRight",
                            "X": 0.5142884850502014,
                            "Y": 0.33167609572410583
                        }
                    ],
                    "Pose": {
                        "Pitch": 15.966927528381348,
                        "Roll": -15.547388076782227,
                        "Yaw": 11.34195613861084
                    },
                    "Quality": {
                        "Brightness": 44.80223083496094,
                        "Sharpness": 99.95819854736328
                    }
                },
                "Index": 1
            },
            "Timestamp": 0
        }.....

    ],
```

```
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

检测个人防护设备

Amazon Rekognition 可以检测图像中人物佩戴的个人防护设备 (PPE)。您可以使用这些信息来改善工作场所的安全实践。例如，您可以使用个人防护设备检测来帮助确定建筑工地上的工人是否佩戴头罩，或者医务人员是否佩戴口罩和手套。下图显示了可以检测到的一些个人防护设备类型。

要检测图像中的个人防护装备，您可以调用 DetectProtectiveEquipmentAPI 并传递输入图像。响应是一个 JSON 结构，其中包含以下内容。

- 在图像中检测到的人。
- 佩戴个人防护设备的身体部位（脸部、头部、左手和右手）。
- 检测到身体部位上的个人防护设备类型（口罩、手套和头罩）。
- 对于检测到的个人防护设备，指明个人防护设备是否覆盖相应身体部位。

返回图像中检测到的人员和个人防护设备位置的边界框。

您还可以要求提供图像中检测到的个人防护设备物品和人员的摘要。有关更多信息，请参阅 汇总图像中检测到的个人防护设备。

> **Note**
>
> Amazon Rekognition 个人防护设备检测不执行人脸识别或人脸比较操作，也无法识别检测到的人员。

个人防护设备的类型

DetectProtectiveEquipment检测以下类型的 PPE。如果您想在图像中检测其他类型的个人防护设备，可以考虑使用 Amazon Rekognition Custom Labels 来训练自定义模型。有关更多信息，请参阅 Amazon Rekognition Custom Labels。

口罩

`DetectProtectiveEquipment` 可以检测常见的口罩，例如外科口罩、N95 口罩和布制口罩。

手套

`DetectProtectiveEquipment` 可以检测手套，例如手术手套和安全手套。

头罩

`DetectProtectiveEquipment` 可以检测安全帽和头盔。

API 表明在图像中检测到头罩、手套或口罩。API 不会返回有关特定遮挡物类型的信息。例如，“外科手套”代表手套类型。

个人防护设备检测置信度

Amazon Rekognition 可预测图像中是否存在个人防护设备、人员和身体部位。API 提供的分数 (50-100) 表示 Amazon Rekognition 对预测准确性的置信度。

> **Note**
>
> 如果您计划使用 DetectProtectiveEquipment 操作来做出影响个人权利、隐私或服务访问权限的决定，我们建议您在采取行动之前将结果交给人类进行审查和验证。

汇总图像中检测到的个人防护设备

您可以选择要求提供图像中检测到的个人防护设备物品和人员的摘要。您可以指定所需防护设备（口罩、手套或头罩）列表和最低可信度阈值（例如 80%）。该响应包括一份按图像标识符 (ID) 分类的综合摘要，其中包括已配备所需个人防护设备的人员、未配备所需个人防护设备的人员以及无法确定的人员。

摘要可以让您快速回答以下问题：有多少人没有戴口罩？或者是否每个人都佩戴了个人防护设备？ 摘要中检测到的每个人都有一个唯一的 ID。您可以使用 ID 查找信息，例如未佩戴个人防护设备者的边界框位置。

> **Note**
>
> ID 是根据每张图像的分析结果随机生成的，在不同图像或同一图像的多次分析中并不一致。

您可以总结口罩、头罩、手套或您选择的组合。要指定所需的个人防护设备类型，请参阅指定汇总要求。您还可以指定最低置信度 (50-100)，必须达到该置信度，检测结果才会被纳入摘要。

有关来自 DetectProtectiveEquipment 的摘要响应的更多信息，请参阅了解回 DetectProtectiveEquipment 应。

教程：创建使用 PPE 检测图像的 AWS Lambda 函数

您可以创建一项 AWS Lambda 功能，用于检测位于 Amazon S3 存储桶中的图像中的个人防护装备 (PPE)。有关此 Java V2 教程，请参阅AWS 文档 SDK 示例 GitHub 存储库。

了解个人防护设备检测 API

以下信息描述了 DetectProtectiveEquipmentAPI。有关代码示例，请参阅 检测图像中的个人防护设备。

提供图片

您可以以图像字节的形式提供输入图像（JPG 或 PNG 格式），也可以引用存储在 Amazon S3 存储桶中的图像。

我们建议使用人物脸部朝向摄像头的图像。

如果您的输入图像未旋转至 0 度，我们建议在提交至 DetectProtectiveEquipment 之前将其旋转至 0 度。JPG 格式的图像可能包含可交换图像文件格式 (Exif) 元数据中的方向信息。您可以使用这些信息编写代码来旋转图像。有关更多信息，请参阅 Exif 版本 2.32。PNG 格式的图像不包含图像方向信息。

要传递来自亚马逊 S3 存储桶的图片，请使用至少具有 AmazonS3 ReadOnlyAccess 权限的用户。使用具有 AmazonRekognitionFullAccess 权限的用户调用 DetectProtectiveEquipment.

在下面的输入 JSON 示例中，图像是通过 Amazon S3 存储桶传递的。有关更多信息，请参阅 使用图像。该示例要求汇总所有个人防护设备类型（头罩、手罩和口罩），最低检测置信度（MinConfidence）为 80%。您应指定一个介于 50-100% 之间的 MinConfidence 值，因为只有当检测置信度介于 50%-100% 之间时，DetectProtectiveEquipment 才会返回预测。如果指定的值小于 50%，则指定值为 50% 的结果相同。有关更多信息，请参阅 指定汇总要求。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "worker.jpg"
        }
    },
    "SummarizationAttributes": {
        "MinConfidence": 80,
        "RequiredEquipmentTypes": [
            "FACE_COVER",
            "HAND_COVER",
            "HEAD_COVER"
        ]
    }
}
```

如果您有大型图像集要做处理，请考虑使用 AWS Batch 在后台分批处理对 DetectProtectiveEquipment 的调用。

指定汇总要求

您可以选择使用 SummarizationAttributes (ProtectiveEquipmentSummarizationAttributes) 输入参数请求图像中检测到的 PPE 类型的摘要信息。

指定要汇总的个人防护设备类型，请使用 RequiredEquipmentTypes 数组字段。在数组中，包含 FACE_COVER、HAND_COVER 或 HEAD_COVER 中的一个或多个。

使用 MinConfidence 字段指定最低检测置信度 (50-100)。摘要不包括人物、身体部位、身体部位覆盖范围和个人防护设备物品，检测到的置信度低于 MinConfidence。

有关来自 DetectProtectiveEquipment 的摘要响应的更多信息，请参阅了解回 DetectProtectiveEquipment 应。

了解回 DetectProtectiveEquipment 应

DetectProtectiveEquipment 返回输入图像中检测到的人的数组。对于每个人，都会返回有关检测到的身体部位和检测到的个人防护设备物品的信息。以下工作人员戴着头罩、手套和口罩的照片的 JSON 如下所示。

在 JSON 中，请注意以下事项。

- 被检测到的人 - Persons 是在图像上检测到的一系列人员（包括未佩戴个人防护设备的人）。DetectProtectiveEquipment 可以在图像中检测到的多达 15 个人身上检测到个人防护设备。数组中的每个ProtectiveEquipmentPerson物体都包含一个人 ID、一个人物边界框、检测到的身体部位和检测到的个人防护用品。ProtectiveEquipmentPerson 中 Confidence 的值表示 Amazon Rekognition 对边界框包含一个人的置信度百分比。
- 身体部位 — BodyParts 是在一个人身上检测到的一系列身体部位 (ProtectiveEquipmentBodyPart)（包括个人防护装备未覆盖的身体部位）。每个 ProtectiveEquipmentBodyPart 包含检测到的身体部位的名称（Name）。DetectProtectEquipment 可以检测脸部、头部、左手和右手这些身体部位。ProtectiveEquipmentBodyPart 中的 Confidence 字段表示 Amazon Rekognition 对身体部位检测精确度的置信度百分比。
- 个人防护设备物品 - ProtectiveEquipmentBodyPart 对象中的数组 EquipmentDetections 包含一组检测到的个人防护设备物品。每个EquipmentDetection对象都包含以下字段。

- Type – 检测到的个人防护设备的类型。
- BoundingBox – 检测到的个人防护设备周围的边界框。
- Confidence – Amazon Rekognition 对边界框中包含检测到的个人防护设备的置信度。
- CoversBodyPart – 表示检测到的个人防护设备是否在相应的身体部位。

该CoversBodyPart字段Value是一个布尔值，用于指示检测到的个人防护装备是否在相应的身体部位。字段Confidence表示对预测的置信度。您可以使用 CoversBodyPart 筛选图像中检测到个人防护设备，但实际上没有出现在人身上的情况。

> **Note**
>
> CoversBodyPart 并不表示或暗示该人已受到防护设备的充分保护，也不表示该人已正确佩戴防护设备。

- 摘要信息 – Summary 包含 SummarizationAttributes 输入参数中指定的摘要信息。有关更多信息，请参阅 指定汇总要求。

  Summary是一个包含以下ProtectiveEquipmentSummary信息的类型的对象。

  - PersonsWithRequiredEquipment— 每个人都符合以下标准 IDs 的人员阵列。
    - 该人穿着 SummarizationAttributes 输入参数中指定的所有个人防护设备。
    - 人员（ProtectiveEquipmentPerson）的 Confidence、身体部位（ProtectiveEquipmentBodyPart）、防护设备（EquipmentDetection）等于或大于指定的最低置信度阈值（MinConfidence）。
    - 所有个人防护设备物品 CoversBodyPart 的值均为真。
  - PersonsWithoutRequiredEquipment— 符合以下条件之一的一系列人员。 IDs
    - 人员（ProtectiveEquipmentPerson）的 Confidence 值、身体部位（ProtectiveEquipmentBodyPart）和身体部位覆盖范围（CoversBodyPart）大于指定的最低置信度阈值（MinConfidence），但该人缺少一个或多个指定的个人防护设备（SummarizationAttributes）。
    - 对于任何Confidence值大于指定的最小置信度阈值（MinConfidence）的指定个人防护设备（SummarizationAttributes），其CoversBodyPart值均为 false。该人还拥有所有指定的个人防护设备（SummarizationAttributes），并且人员（ProtectiveEquipmentPerson）、身体部位（ProtectiveEquipmentBodyPart）和防护设备（EquipmentDetection）的Confidence值大于或等于最低置信阈值（MinConfidence）。

- PersonsIndeterminate— 检测到的人员 IDs 数组，其中人员 (ProtectiveEquipmentPerson)、身体部位 (ProtectiveEquipmentBodyPart)、防护装备 (EquipmentDetection) 或CoversBodyPart布尔值低于指定的最小置信阈值 (MinConfidence)。Confidence

使用数组大小来获取特定摘要的计数。例如，PersonsWithRequiredEquipment 的大小告诉您检测到佩戴指定类型个人防护设备的人数。

您可以使用人员 ID 来查找有关某人的更多信息，例如该人的边界框位置。人员 ID 映射到Persons（ProtectiveEquipmentPerson 的数组）中返回的 ProtectiveEquipmentPerson 对象的 ID 字段。然后，您可以从相应的 ProtectiveEquipmentPerson 对象中获取边界框和其他信息。

```
{
   "ProtectiveEquipmentModelVersion": "1.0",
   "Persons": [
       {
           "BodyParts": [
               {
                   "Name": "FACE",
                   "Confidence": 99.99861145019531,
                   "EquipmentDetections": [
                       {
                           "BoundingBox": {
                               "Width": 0.14528800547122955,
                               "Height": 0.14956723153591156,
                               "Left": 0.4363413453102112,
                               "Top": 0.34203192591667175
                           },
                           "Confidence": 99.90001678466797,
                           "Type": "FACE_COVER",
                           "CoversBodyPart": {
                               "Confidence": 98.0676498413086,
                               "Value": true
                           }
                       }
                   ]
               },
               {
                   "Name": "LEFT_HAND",
```

```
                    "Confidence": 96.9786376953125,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
                                "Width": 0.14495663344860077,
                                "Height": 0.12936046719551086,
                                "Left": 0.5114737153053284,
                                "Top": 0.5744519829750061
                            },
                            "Confidence": 83.72270965576172,
                            "Type": "HAND_COVER",
                            "CoversBodyPart": {
                                "Confidence": 96.9288558959961,
                                "Value": true
                            }
                        }
                    ]
                },
                {
                    "Name": "RIGHT_HAND",
                    "Confidence": 99.82939147949219,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
                                "Width": 0.20971858501434326,
                                "Height": 0.20528452098369598,
                                "Left": 0.2711356580257416,
                                "Top": 0.6750612258911133
                            },
                            "Confidence": 95.70789337158203,
                            "Type": "HAND_COVER",
                            "CoversBodyPart": {
                                "Confidence": 99.85433197021484,
                                "Value": true
                            }
                        }
                    ]
                },
                {
                    "Name": "HEAD",
                    "Confidence": 99.9999008178711,
                    "EquipmentDetections": [
                        {
                            "BoundingBox": {
```

```
                                "Width": 0.24350935220718384,
                                "Height": 0.34623199701309204,
                                "Left": 0.43011072278022766,
                                "Top": 0.01103297434747219
                            },
                            "Confidence": 83.88762664794922,
                            "Type": "HEAD_COVER",
                            "CoversBodyPart": {
                                "Confidence": 99.96485900878906,
                                "Value": true
                            }
                        }
                    ]
                }
            ],
            "BoundingBox": {
                "Width": 0.7403100728988647,
                "Height": 0.9412225484848022,
                "Left": 0.02214839495718479,
                "Top": 0.03134796395897865
            },
            "Confidence": 99.98855590820312,
            "Id": 0
        }
    ],
    "Summary": {
        "PersonsWithRequiredEquipment": [
            0
        ],
        "PersonsWithoutRequiredEquipment": [],
        "PersonsIndeterminate": []
    }
}
```

检测图像中的个人防护设备

要检测图像中人物身上的个人防护装备 (PPE)，请使用DetectProtectiveEquipment非存储 API 操作。

您可以使用 AWS 开发工具包或 AWS Command Line Interface （ AWS CLI ），以图像字节数组（base64 编码的图像字节）或 Amazon S3 对象的形式提供输入图像。这些示例使用存储在 Amazon S3 存储桶中的一个图像。有关更多信息，请参阅 使用图像。

检测图像中人物身上的个人防护设备

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将（包含一个或多个佩戴个人防护设备的人）的图像上传到您的 S3 存储桶。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 DetectProtectiveEquipment 操作。有关显示图像中的边界框的信息，请参见显示边界框。

   Java

   此示例显示有关在图像中检测到的人身上检测到的个人防护设备物品的信息。

   将amzn-s3-demo-bucket的值更改为包含图像的 Amazon S3 存储桶的名称。将photo的值更改为图像文件名。

```
//Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;
import
 com.amazonaws.services.rekognition.model.ProtectiveEquipmentSummarizationAttributes;

import java.util.List;
import com.amazonaws.services.rekognition.model.BoundingBox;
```

```
import
 com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;


public class DetectPPE {

    public static void main(String[] args) throws Exception {

        String photo = "photo";
        String bucket = "bucket";


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        ProtectiveEquipmentSummarizationAttributes summaryAttributes = new
 ProtectiveEquipmentSummarizationAttributes()
                .withMinConfidence(80F)
                .withRequiredEquipmentTypes("FACE_COVER", "HAND_COVER",
 "HEAD_COVER");

        DetectProtectiveEquipmentRequest request = new
 DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName(photo).withBucket(bucket)))
                .withSummarizationAttributes(summaryAttributes);

        try {
            System.out.println("Detected PPE for people in image " + photo);
            System.out.println("Detected people\n---------------");
            DetectProtectiveEquipmentResult result =
 rekognitionClient.detectProtectiveEquipment(request);


            List <ProtectiveEquipmentPerson> persons = result.getPersons();


            for (ProtectiveEquipmentPerson person: persons) {
                System.out.println("ID: " + person.getId());
                List<ProtectiveEquipmentBodyPart>
 bodyParts=person.getBodyParts();
```

```
                if (bodyParts.isEmpty()){
                    System.out.println("\tNo body parts detected");
                } else
                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                        System.out.println("\t" + bodyPart.getName() + ".
Confidence: " + bodyPart.getConfidence().toString());



                        List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                        if (equipmentDetections.isEmpty()){
                            System.out.println("\t\tNo PPE Detected on " +
bodyPart.getName());

                        }
                        else {
                            for (EquipmentDetection item: equipmentDetections) {
                                System.out.println("\t\tItem: " + item.getType()
+ ". Confidence: " + item.getConfidence().toString());
                                System.out.println("\t\tCovers body part: "
                                        +
item.getCoversBodyPart().getValue().toString() + ". Confidence: " +
item.getCoversBodyPart().getConfidence().toString());

                                System.out.println("\t\tBounding Box");
                                BoundingBox box =item.getBoundingBox();

                                System.out.println("\t\tLeft: "
+box.getLeft().toString());
                                System.out.println("\t\tTop: " +
box.getTop().toString());
                                System.out.println("\t\tWidth: " +
box.getWidth().toString());
                                System.out.println("\t\tHeight: " +
box.getHeight().toString());
                                System.out.println("\t\tConfidence: " +
item.getConfidence().toString());
                                System.out.println();
                            }
                        }

                    }
```

```
            }
            System.out.println("Person ID Summary\n-----------------");

            //List<Integer> list=;
            DisplaySummary("With required equipment",
 result.getSummary().getPersonsWithRequiredEquipment());
            DisplaySummary("Without required equipment",
 result.getSummary().getPersonsWithoutRequiredEquipment());
            DisplaySummary("Indeterminate",
 result.getSummary().getPersonsIndeterminate());


        } catch(AmazonRekognitionException e) {
            e.printStackTrace();
        }
    }
    static void DisplaySummary(String summaryType,List<Integer> idList)
    {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        }
        else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                }
                else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }

        System.out.println();


    }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.detect_ppe.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_ppe.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectPPE {

 public static void main(String[] args) {

     final String usage = "\n" +
```

```
        "Usage: " +
        "   <sourceImage> <bucketName>\n\n" +
        "Where:\n" +
        "   sourceImage - The name of the image in an Amazon S3 bucket (for
 example, people.png). \n\n" +
        "   bucketName - The name of the Amazon S3 bucket (for example, amzn-
s3-demo-bucket). \n\n";

    if (args.length != 2) {
         System.out.println(usage);
         System.exit(1);
    }

    String sourceImage = args[0];
    String bucketName = args[1];
    Region region = Region.US_WEST_2;
    S3Client s3 = S3Client.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    displayGear(s3, rekClient, sourceImage, bucketName) ;
    s3.close();
    rekClient.close();
    System.out.println("This example is done!");
}

// snippet-start:[rekognition.java2.detect_ppe.main]
public static void displayGear(S3Client s3,
                               RekognitionClient rekClient,
                               String sourceImage,
                               String bucketName) {

    byte[] data = getObjectBytes (s3, bucketName, sourceImage);
    InputStream is = new ByteArrayInputStream(data);

    try {
        ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
 ProtectiveEquipmentSummarizationAttributes.builder()
```

```
            .minConfidence(80F)
            .requiredEquipmentTypesWithStrings("FACE_COVER", "HAND_COVER",
"HEAD_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectProtectiveEquipmentRequest request =
DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
rekClient.detectProtectiveEquipment(request);
        List<ProtectiveEquipmentPerson> persons = result.persons();
        for (ProtectiveEquipmentPerson person: persons) {
            System.out.println("ID: " + person.id());
            List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
            if (bodyParts.isEmpty()){
                System.out.println("\tNo body parts detected");
            } else
                for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                    System.out.println("\t" + bodyPart.name() + ". Confidence:
" + bodyPart.confidence().toString());
                    List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();

                    if (equipmentDetections.isEmpty()){
                        System.out.println("\t\tNo PPE Detected on " +
bodyPart.name());
                    } else {
                        for (EquipmentDetection item: equipmentDetections) {
                            System.out.println("\t\tItem: " + item.type() + ".
Confidence: " + item.confidence().toString());
                            System.out.println("\t\tCovers body part: "
                                    + item.coversBodyPart().value().toString()
+ ". Confidence: " + item.coversBodyPart().confidence().toString());

                            System.out.println("\t\tBounding Box");
```

```
                            BoundingBox box =item.boundingBox();
                            System.out.println("\t\tLeft: "
+box.left().toString());
                            System.out.println("\t\tTop: " +
box.top().toString());
                            System.out.println("\t\tWidth: " +
box.width().toString());
                            System.out.println("\t\tHeight: " +
box.height().toString());
                            System.out.println("\t\tConfidence: " +
item.confidence().toString());
                            System.out.println();
                        }
                    }
                }
        }
        System.out.println("Person ID Summary\n-----------------");

        displaySummary("With required equipment",
result.summary().personsWithRequiredEquipment());
        displaySummary("Without required equipment",
result.summary().personsWithoutRequiredEquipment());
        displaySummary("Indeterminate",
result.summary().personsIndeterminate());

    } catch (RekognitionException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

public static byte[] getObjectBytes (S3Client s3, String bucketName, String
keyName) {

    try {
        GetObjectRequest objectRequest = GetObjectRequest
            .builder()
            .key(keyName)
            .bucket(bucketName)
            .build();

        ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
        return objectBytes.asByteArray();
```

```
        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    static void displaySummary(String summaryType,List<Integer> idList) {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        } else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                } else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }
        System.out.println();
    }
    // snippet-end:[rekognition.java2.detect_ppe.main]
}
```

AWS CLI

此 AWS CLI 命令请求 PPE 摘要并显示 `detect-protective-equipment` CLI 操作的 JSON 输出。

将`amzn-s3-demo-bucket`更改为包含图像的 Amazon S3 存储桶的名称。将 `input.jpg` 更改为要使用的图像的名称。

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}" \
  --summarization-attributes
 "MinConfidence=80,RequiredEquipmentTypes=['FACE_COVER','HAND_COVER','HEAD_COVER']"
```

此 AWS CLI 命令显示 `detect-protective-equipment` CLI 操作的 JSON 输出。

将amzn-s3-demo-bucket更改为包含图像的 Amazon S3 存储桶的名称。将 input.jpg 更改为要使用的图像的名称。

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

Python

此示例显示有关在图像中检测到的人身上检测到的个人防护设备物品的信息。

将amzn-s3-demo-bucket的值更改为包含图像的 Amazon S3 存储桶的名称。将photo的值更改为图像文件名。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def detect_ppe(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.detect_protective_equipment(Image={'S3Object': {'Bucket':
 bucket, 'Name': photo}},

 SummarizationAttributes={'MinConfidence': 80,

 'RequiredEquipmentTypes': ['FACE_COVER',

                     'HAND_COVER',

                     'HEAD_COVER']})

    print('Detected PPE for people in image ' + photo)
    print('\nDetected people\n---------------')
    for person in response['Persons']:

        print('Person ID: ' + str(person['Id']))
```

```
        print('Body Parts\n----------')
        body_parts = person['BodyParts']
        if len(body_parts) == 0:
            print('No body parts found')
        else:
            for body_part in body_parts:
                print('\t' + body_part['Name'] + '\n\t\tConfidence: ' +
 str(body_part['Confidence']))
                print('\n\t\tDetected PPE\n\t\t------------')
                ppe_items = body_part['EquipmentDetections']
                if len(ppe_items) == 0:
                    print('\t\tNo PPE detected on ' + body_part['Name'])
                else:
                    for ppe_item in ppe_items:
                        print('\t\t' + ppe_item['Type'] + '\n\t\t\tConfidence: '
 + str(ppe_item['Confidence']))
                        print('\t\tCovers body part: ' + str(
                            ppe_item['CoversBodyPart']['Value']) + '\n\t\t
\tConfidence: ' + str(
                            ppe_item['CoversBodyPart']['Confidence']))
                        print('\t\tBounding Box:')
                        print('\t\t\tTop: ' + str(ppe_item['BoundingBox']
['Top']))
                        print('\t\t\tLeft: ' + str(ppe_item['BoundingBox']
['Left']))
                        print('\t\t\tWidth: ' + str(ppe_item['BoundingBox']
['Width']))
                        print('\t\t\tHeight: ' + str(ppe_item['BoundingBox']
['Height']))
                        print('\t\t\tConfidence: ' +
 str(ppe_item['Confidence']))
            print()
        print()

    print('Person ID Summary\n----------------')
    display_summary('With required equipment', response['Summary']
['PersonsWithRequiredEquipment'])
    display_summary('Without required equipment', response['Summary']
['PersonsWithoutRequiredEquipment'])
    display_summary('Indeterminate', response['Summary']
['PersonsIndeterminate'])

    print()
    return len(response['Persons'])
```

```
# Display summary information for supplied summary.
def display_summary(summary_type, summary):
    print(summary_type + '\n\tIDs: ', end='')
    if (len(summary) == 0):
        print('None')
    else:
        for num, id in enumerate(summary, start=0):
            if num == len(summary) - 1:
                print(id)
            else:
                print(str(id) + ', ', end='')

def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    person_count = detect_ppe(photo, bucket)
    print("Persons detected: " + str(person_count))

if __name__ == "__main__":
    main()
```

示例：在口罩周围绘制边界框

以下示例向您展示了如何在人身上检测到的口罩周围绘制边界框。有关使用 AWS Lambda 和 Amazon DynamoDB 的示例，请参阅文档软件开发工具包示例AWS 存储库。 GitHub

要检测面罩，您可以使用DetectProtectiveEquipment非存储 API 操作。图像是从本地文件系统加载的。您可以为 `DetectProtectiveEquipment` 提供输入图像作为图像字节数组（base64 编码的图像字节）。有关更多信息，请参阅 使用图像。

该示例在检测到的口罩周围显示一个边界框。如果口罩完全覆盖身体部位，则边界框为绿色。否则，将显示一个红色的边界框。作为警告，如果检测置信度低于指定的置信度值，则口罩边界框内会显示一个黄色的边界框。如果未检测到口罩，则会在该人周围画一个红色的边界框。

图像输出类似于以下内容。

在检测到的口罩上显示边界框

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 DetectProtectiveEquipment 操作。有关显示图像中的边界框的信息，请参见显示边界框。

   Java

   在函数 main 中，更改以下内容：

   - photo的值为本地图像文件（PNG 或 JPEG）的路径和文件名。

- confidence的值达到所需的置信度水平 (50-100)。

```
//Loads images, detects faces and draws bounding boxes.Determines exif
 orientation, if necessary.
package com.amazonaws.samples;


import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.BoundingBox;
import
 com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;

// Calls DetectFaces and displays a bounding box around each detected image.
public class PPEBoundingBox extends JPanel {

    private static final long serialVersionUID = 1L;

    BufferedImage image;
    static int scale;
    DetectProtectiveEquipmentResult result;
```

```
    float confidence=80;

    public PPEBoundingBox(DetectProtectiveEquipmentResult ppeResult,
BufferedImage bufImage, float requiredConfidence) throws Exception {
        super();
        scale = 2; // increase to shrink image size.

        result = ppeResult;
        image = bufImage;

        confidence=requiredConfidence;
    }
    // Draws the bounding box around the detected faces.
    public void paintComponent(Graphics g) {
        float left = 0;
        float top = 0;
        int height = image.getHeight(this);
        int width = image.getWidth(this);
        int offset=20;

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
        g2d.setColor(new Color(0, 212, 0));

        // Iterate through detected persons and display bounding boxes.
        List<ProtectiveEquipmentPerson> persons = result.getPersons();

        for (ProtectiveEquipmentPerson person: persons) {
            BoundingBox boxPerson = person.getBoundingBox();
            left = width * boxPerson.getLeft();
            top = height * boxPerson.getTop();
            Boolean foundMask=false;

            List<ProtectiveEquipmentBodyPart> bodyParts=person.getBodyParts();

            if (bodyParts.isEmpty()==false)
             {
                    //body parts detected

                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
```

```
                        List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                        for (EquipmentDetection item: equipmentDetections) {

                            if (item.getType().contentEquals("FACE_COVER"))
                            {
                                // Draw green or red bounding box depending on
mask coverage.
                                foundMask=true;
                                BoundingBox box =item.getBoundingBox();
                                left = width * box.getLeft();
                                top = height * box.getTop();
                                Color maskColor=new Color( 0, 212, 0);

                                if (item.getCoversBodyPart().getValue()==false)
{
                                    // red bounding box
                                    maskColor=new Color( 255, 0, 0);
                                }
                                g2d.setColor(maskColor);
                                g2d.drawRect(Math.round(left / scale),
Math.round(top / scale),
                                        Math.round((width * box.getWidth()) /
scale), Math.round((height * box.getHeight())) / scale);

                                // Check confidence is > supplied confidence.
                                if (item.getCoversBodyPart().getConfidence()<
confidence)
                                {
                                    // Draw a yellow bounding box inside face
mask bounding box
                                    maskColor=new Color( 255, 255, 0);
                                    g2d.setColor(maskColor);
                                    g2d.drawRect(Math.round((left + offset) /
scale),
                                            Math.round((top + offset) / scale),
                                            Math.round((width *
box.getWidth())- (offset * 2 ))/ scale,
                                            Math.round((height *
box.getHeight()) -( offset* 2)) / scale);
                                }

                            }
```

```
                        }

                    }

                }

            // Didn't find a mask, so draw person bounding box red
            if (foundMask==false) {

                left = width * boxPerson.getLeft();
                top = height * boxPerson.getTop();
                g2d.setColor(new Color(255, 0, 0));
                g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                        Math.round(((width) * boxPerson.getWidth()) / scale),
Math.round((height * boxPerson.getHeight())) / scale);
            }
         }

   }


   public static void main(String arg[]) throws Exception {

       String photo = "photo";

       float confidence =80;


       int height = 0;
       int width = 0;

       BufferedImage image = null;
       ByteBuffer imageBytes;

       // Get image bytes for call to DetectProtectiveEquipment
       try (InputStream inputStream = new FileInputStream(new File(photo))) {
           imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
       }

       //Get image for display
       InputStream imageBytesStream;
       imageBytesStream = new ByteArrayInputStream(imageBytes.array());

       ByteArrayOutputStream baos = new ByteArrayOutputStream();
```

```
        image=ImageIO.read(imageBytesStream);
        ImageIO.write(image, "jpg", baos);
        width = image.getWidth();
        height = image.getHeight();

        //Get Rekognition client
        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


        // Call DetectProtectiveEquipment
        DetectProtectiveEquipmentRequest request = new
 DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withBytes(imageBytes));

        DetectProtectiveEquipmentResult result =
 rekognitionClient.detectProtectiveEquipment(request);


        // Create frame and panel.
        JFrame frame = new JFrame("Detect PPE");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        PPEBoundingBox panel = new PPEBoundingBox(result, image, confidence);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.List;
import javax.imageio.ImageIO;
```

```
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
//snippet-end:[rekognition.java2.display_mask.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class PPEBoundingBoxFrame extends JPanel {

 DetectProtectiveEquipmentResponse result;
 static BufferedImage image;
 static int scale;
 float confidence;

 public static void main(String[] args) throws Exception {
```

```
    final String usage = "\n" +
        "Usage: " +
        "   <sourceImage> <bucketName>\n\n" +
        "Where:\n" +
        "   sourceImage - The name of the image in an Amazon S3 bucket that
 shows a person wearing a mask (for example, masks.png). \n\n" +
        "   bucketName - The name of the Amazon S3 bucket (for example, amzn-
s3-demo-bucket). \n\n";

    if (args.length != 2) {
        System.out.println(usage);
        System.exit(1);
    }

    String sourceImage = args[0];
    String bucketName = args[1];
    Region region = Region.US_EAST_1;
    S3Client s3 = S3Client.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    displayGear(s3, rekClient, sourceImage, bucketName);
    s3.close();
    rekClient.close();
 }

 // snippet-start:[rekognition.java2.display_mask.main]
public static void displayGear(S3Client s3,
                                   RekognitionClient rekClient,
                                   String sourceImage,
                                   String bucketName) {
    float confidence = 80;
    byte[] data = getObjectBytes(s3, bucketName, sourceImage);
    InputStream is = new ByteArrayInputStream(data);

    try {
        ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
 ProtectiveEquipmentSummarizationAttributes.builder()
```

```
            .minConfidence(70F)
            .requiredEquipmentTypesWithStrings("FACE_COVER")
            .build();

        SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
        image = ImageIO.read(sourceBytes.asInputStream());

        // Create an Image object for the source image.
        software.amazon.awssdk.services.rekognition.model.Image souImage =
 Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectProtectiveEquipmentRequest request =
 DetectProtectiveEquipmentRequest.builder()
            .image(souImage)
            .summarizationAttributes(summarizationAttributes)
            .build();

        DetectProtectiveEquipmentResponse result =
 rekClient.detectProtectiveEquipment(request);
        JFrame frame = new JFrame("Detect PPE");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        PPEBoundingBoxFrame panel = new PPEBoundingBoxFrame(result, image,
 confidence);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    } catch (RekognitionException e) {
        e.printStackTrace();
        System.exit(1);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

 public static byte[] getObjectBytes (S3Client s3, String bucketName, String
 keyName) {

     try {
         GetObjectRequest objectRequest = GetObjectRequest
```

```
            .builder()
            .key(keyName)
            .bucket(bucketName)
            .build();

        ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
        return objectBytes.asByteArray();

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
 }

public PPEBoundingBoxFrame(DetectProtectiveEquipmentResponse ppeResult,
BufferedImage bufImage, float requiredConfidence) {
    super();
    scale = 1; // increase to shrink image size.
    result = ppeResult;
    image = bufImage;
    confidence=requiredConfidence;
}

// Draws the bounding box around the detected masks.
public void paintComponent(Graphics g) {
    float left = 0;
    float top = 0;
    int height = image.getHeight(this);
    int width = image.getWidth(this);
    int offset=20;

    Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

    // Draw the image.
    g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
    g2d.setColor(new Color(0, 212, 0));

    // Iterate through detected persons and display bounding boxes.
    List<ProtectiveEquipmentPerson> persons = result.persons();
    for (ProtectiveEquipmentPerson person: persons) {

        List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
```

```
        if (!bodyParts.isEmpty()){
            for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                List<EquipmentDetection>
 equipmentDetections=bodyPart.equipmentDetections();
                for (EquipmentDetection item: equipmentDetections) {

                    String myType = item.type().toString();
                    if (myType.compareTo("FACE_COVER") ==0) {

                        // Draw green bounding box depending on mask coverage.
                        BoundingBox box =item.boundingBox();
                        left = width * box.left();
                        top = height * box.top();
                        Color maskColor=new Color( 0, 212, 0);

                        if (item.coversBodyPart().equals(false)) {
                            // red bounding box.
                            maskColor=new Color( 255, 0, 0);
                        }
                        g2d.setColor(maskColor);
                        g2d.drawRect(Math.round(left / scale), Math.round(top /
 scale),
                                Math.round((width * box.width()) / scale),
 Math.round((height * box.height())) / scale);

                        // Check confidence is > supplied confidence.
                        if (item.coversBodyPart().confidence() < confidence) {
                            // Draw a yellow bounding box inside face mask
 bounding box.
                            maskColor=new Color( 255, 255, 0);
                            g2d.setColor(maskColor);
                            g2d.drawRect(Math.round((left + offset) / scale),
                                    Math.round((top + offset) / scale),
                                    Math.round((width * box.width())- (offset *
 2 ))/ scale,
                                    Math.round((height * box.height()) -
( offset* 2)) / scale);
                        }
                    }
                }
            }
        }
    }
 }
```

```
 // snippet-end:[rekognition.java2.display_mask.main]
}
```

Python

在函数 `main` 中，更改以下内容：

- `photo`的值为本地图像文件（PNG 或 JPEG）的路径和文件名。
- `confidence`的值达到所需的置信度水平 (50-100)。
- 将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
#Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import io
from PIL import Image, ImageDraw, ExifTags, ImageColor

def detect_ppe(photo, confidence):

    fill_green='#00d400'
    fill_red='#ff0000'
    fill_yellow='#ffff00'
    line_width=3

    #open image and get image data from stream.
    image = Image.open(open(photo,'rb'))
    stream = io.BytesIO()
    image.save(stream, format=image.format)
    image_binary = stream.getvalue()
    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)

    client=boto3.client('rekognition')

    response = client.detect_protective_equipment(Image={'Bytes': image_binary})

    for person in response['Persons']:

        found_mask=False
```

```
        for body_part in person['BodyParts']:
            ppe_items = body_part['EquipmentDetections']

            for ppe_item in ppe_items:
                #found a mask
                if ppe_item['Type'] == 'FACE_COVER':
                    fill_color=fill_green
                    found_mask=True
                    # check if mask covers face
                    if ppe_item['CoversBodyPart']['Value'] == False:
                        fill_color=fill='#ff0000'
                    # draw bounding box around mask
                    box = ppe_item['BoundingBox']
                    left = imgWidth * box['Left']
                    top = imgHeight * box['Top']
                    width = imgWidth * box['Width']
                    height = imgHeight * box['Height']
                    points = (
                            (left,top),
                            (left + width, top),
                            (left + width, top + height),
                            (left , top + height),
                            (left, top)
                        )
                    draw.line(points, fill=fill_color, width=line_width)

                     # Check if confidence is lower than supplied value
                    if ppe_item['CoversBodyPart']['Confidence'] < confidence:
                        #draw warning yellow bounding box within face mask
bounding box
                        offset=line_width+ line_width
                        points = (
                                    (left+offset,top + offset),
                                    (left + width-offset, top+offset),
                                    ((left) + (width-offset), (top-offset) +
(height)),
                                    (left+ offset , (top) + (height -offset)),
                                    (left + offset, top + offset)
                                )
                        draw.line(points, fill=fill_yellow, width=line_width)

        if found_mask==False:
            # no face mask found so draw red bounding box around body
```

```
            box = person['BoundingBox']
            left = imgWidth * box['Left']
            top = imgHeight * box['Top']
            width = imgWidth * box['Width']
            height = imgHeight * box['Height']
            points = (
                (left,top),
                (left + width, top),
                (left + width, top + height),
                (left , top + height),
                (left, top)
                )
            draw.line(points, fill=fill_red, width=line_width)

    image.show()

def main():
    photo='photo'
    confidence=80
    detect_ppe(photo, confidence)

if __name__ == "__main__":
    main()
```

CLI

在以下 CLI 示例中，更改下面列出的参数的值：

- photo的值为本地图像文件（PNG 或 JPEG）的路径和文件名。
- confidence的值达到所需的置信度水平 (50-100)。
- 将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
                                aws rekognition detect-protective-equipment
 --image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" --
profile profile-name \
                                --summarization-attributes
 "{"MinConfidence":MinConfidenceNumber,"RequiredEquipmentTypes":["FACE_COVER"]}"
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
                                aws rekognition detect-protective-equipment  --
image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"image-name
\"}}" \
                                --profile profile-name --summarization-
attributes "{\"MinConfidence\":MinConfidenceNumber,\"RequiredEquipmentTypes\":
[\"FACE_COVER\"]}"
```

识别名人

借助 Amazon Rekognition，客户可以轻松地使用机器学习自动识别图像和视频中成千上万的知名人物。名人识别 API 提供的元数据大大减少了标记内容所需的重复手动工作，使其易于搜索。

图像和视频内容的迅速激增意味着媒体公司往往难以大规模组织、搜索和利用其媒体目录。新闻频道和体育广播公司通常需要快速找到图像和视频，以便对时事做出反应并制作相关节目。元数据不足会使这些任务变得困难，但是借助 Amazon Rekognition，您可以自动标记大量新内容或存档内容，以便可以轻松搜索各种国际知名人士，例如演员、运动员和在线内容创作者。

Amazon Rekognition 名人识别旨在仅用于您预计图像或视频中可能有知名名人的情况。有关非名人面部识别的信息，请参阅在集合中搜索人脸。

> **Note**
>
> 如果你是名人并且不想参与此功能，请联系AWS 支持团队或发送电子邮件至 <rekognition-celebrity-opt-out@amazon .com>。

主题

- 名人识别与人脸搜索比较
- 识别图像中的名人
- 识别存储视频中的名人
- 获取有关名人的信息

名人识别与人脸搜索比较

Amazon Rekognition 同时提供了名人识别和人脸识别功能。这些功能在使用案例和最佳实践方面有一些重要差别。

名人识别功能经过预训练，能够识别体育、媒体、政治和商业等领域成千上万的受欢迎人士。此功能旨在帮助您搜索大量的图像或视频，以识别可能包含某个特定名人的一小部分图像或视频。它并不是为了将人脸与不同的非名人人士匹配。在名人匹配的准确性非常重要的情况下，我们还建议使用人工操作员查看这少量的标记内容，以帮助确保高准确性并正确应用人工判断。名人识别不应以可能对公民自由产生负面影响的方式使用。

相反，人脸识别是一种更常用的功能，它支持您使用自己的人脸向量创建自己的人脸集合，以验证身份或搜索任何人，而不仅仅是名人。人脸识别可用于楼宇门禁、公共安全和社交媒体的身份验证等应用。在所有这些情况下，建议您在匹配准确性很重要的情况下采用最佳实践、适当的置信度阈值（包括 99% 的公共安全使用案例）和人工审查。

有关更多信息，请参阅 在集合中搜索人脸。

识别图像中的名人

要识别图像中的名人并获取有关所识别名人的其他信息，请使用 RecognizeCelebrities 非存储 API 操作。例如，在社交媒体或新闻和娱乐界，信息收集时间紧张，您可使用 `RecognizeCelebrities` 操作识别一张图像中多达 64 位名人并返回指向名人网页（如有）的链接。Amazon Rekognition 不会记住它在哪一张图像中检测到过名人。您的应用程序必须存储此信息。

如果您尚未存储 `RecognizeCelebrities` 返回的某位名人的其他信息，并且您希望避免重新分析图像以获取这些信息，请使用 GetCelebrityInfo。要调用 `GetCelebrityInfo`，您需要 Amazon Rekognition 分配给每位名人的唯一标识符。该标识符将作为针对在图像中识别的每位名人的 `RecognizeCelebrities` 响应的一部分返回。

如果您有大型图像集要做名人识别处理，请考虑使用 AWS Batch 在后台分批处理对 `RecognizeCelebrities` 的调用。向收藏夹中添加新图片时，您可以使用 AWS Lambda 函数识别名人，方法是在图片上传到 S3 存储桶时调用`RecognizeCelebrities`。

正在呼叫 RecognizeCelebrities

您可以使用 () 或 AWS 软件开发工具包以图像字节数组（base64 编码的图像字节）或 Amazon S3 对象的 AWS Command Line Interface 形式提供输入图像。AWS CLI在此 AWS CLI 过程中，您将.jpg 或.png 格式的图像上传到 S3 存储桶。在 AWS SDK 过程中，您使用的是从本地文件系统加载的图像。有关图像建议的信息，请参阅使用图像。

要运行此过程，您需要一个包含一张或多张名人人脸的图像文件。

识别图像中的名人

1. 如果您尚未执行以下操作，请：

   a. 使用 `AmazonRekognitionFullAccess` 和 `AmazonS3ReadOnlyAccess` 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 RecognizeCelebrities 操作。

Java

此示例显示与图像中检测到的名人相关的信息。

将 photo 的值更改为包含一张或多张名人人脸的图像的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;
import java.util.List;


public class RecognizeCelebrities {

   public static void main(String[] args) {
      String photo = "moviestars.jpg";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ByteBuffer imageBytes=null;
      try (InputStream inputStream = new FileInputStream(new File(photo))) {
         imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load file " + photo);
```

```
            System.exit(1);
        }


        RecognizeCelebritiesRequest request = new RecognizeCelebritiesRequest()
           .withImage(new Image()
           .withBytes(imageBytes));

        System.out.println("Looking for celebrities in image " + photo + "\n");

        RecognizeCelebritiesResult
 result=rekognitionClient.recognizeCelebrities(request);

        //Display recognized celebrity information
        List<Celebrity> celebs=result.getCelebrityFaces();
        System.out.println(celebs.size() + " celebrity(s) were recognized.\n");

        for (Celebrity celebrity: celebs) {
            System.out.println("Celebrity recognized: " + celebrity.getName());
            System.out.println("Celebrity ID: " + celebrity.getId());
            BoundingBox boundingBox=celebrity.getFace().getBoundingBox();
            System.out.println("position: " +
               boundingBox.getLeft().toString() + " " +
               boundingBox.getTop().toString());
            System.out.println("Further information (if available):");
            for (String url: celebrity.getUrls()){
               System.out.println(url);
            }
            System.out.println();
         }
         System.out.println(result.getUnrecognizedFaces().size() + " face(s) were
 unrecognized.");
      }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.recognize_celebs.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesRequest;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Celebrity;
//snippet-end:[rekognition.java2.recognize_celebs.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class RecognizeCelebrities {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length != 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
```

```
        .build();

    System.out.println("Locating celebrities in " + sourceImage);
    recognizeAllCelebrities(rekClient, sourceImage);
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_celebs.main]
public static void recognizeAllCelebrities(RekognitionClient rekClient, String
sourceImage) {

    try {
        InputStream sourceStream = new FileInputStream(sourceImage);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
        Image souImage = Image.builder()
            .bytes(sourceBytes)
            .build();

        RecognizeCelebritiesRequest request =
RecognizeCelebritiesRequest.builder()
            .image(souImage)
            .build();

        RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request) ;
        List<Celebrity> celebs=result.celebrityFaces();
        System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
        for (Celebrity celebrity: celebs) {
            System.out.println("Celebrity recognized: " + celebrity.name());
            System.out.println("Celebrity ID: " + celebrity.id());

            System.out.println("Further information (if available):");
            for (String url: celebrity.urls()){
                System.out.println(url);
            }
            System.out.println();
        }
        System.out.println(result.unrecognizedFaces().size() + " face(s) were
unrecognized.");

    } catch (RekognitionException | FileNotFoundException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
```

```
 }
 // snippet-end:[rekognition.java2.recognize_celebs.main]
}
```

AWS CLI

此 AWS CLI 命令显示 recognize-celebrities CLI 操作的 JSON 输出。

将amzn-s3-demo-bucket更改为包含图像的 Amazon S3 存储桶的名称。将 input.jpg 更改为包含一张或多张名人人脸的图像的文件名。

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition recognize-celebrities \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
                                    aws rekognition recognize-celebrities --
image \
                                    "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile profile-name
```

Python

此示例显示与图像中检测到的名人相关的信息。

将 photo 的值更改为包含一张或多张名人人脸的图像的路径和文件名。

将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def recognize_celebrities(photo):
```

```
    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    with open(photo, 'rb') as image:
        response = client.recognize_celebrities(Image={'Bytes': image.read()})

    print('Detected faces for ' + photo)
    for celebrity in response['CelebrityFaces']:
        print('Name: ' + celebrity['Name'])
        print('Id: ' + celebrity['Id'])
        print('KnownGender: ' + celebrity['KnownGender']['Type'])
        print('Smile: ' + str(celebrity['Face']['Smile']['Value']))
        print('Position:')
        print('   Left: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Height']))
        print('   Top: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Top']))
        print('Info')
        for url in celebrity['Urls']:
            print('   ' + url)
        print()
    return len(response['CelebrityFaces'])

def main():
    photo = 'photo-name'
    celeb_count = recognize_celebrities(photo)
    print("Celebrities detected: " + str(celeb_count))

if __name__ == "__main__":
    main()
```

Node.Js

此示例显示与图像中检测到的名人相关的信息。

将 photo 的值更改为包含一张或多张名人人脸的图像的路径和文件名。将amzn-s3-demo-bucket的值更改为包含所提供图像文件的 S3 存储桶的名称。将REGION的值更改为与您的用户关联的区域名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
// Import required AWS SDK clients and commands for Node.js
import { RecognizeCelebritiesCommand } from  "@aws-sdk/client-rekognition";
```

```
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
const profileName = "profile-name";

// Create SNS service object.
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

const bucket = 'bucket-name'
const photo = 'photo-name'

// Set params
const params = {
    Image: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
  }

const recognize_celebrity = async() => {
    try {
        const response = await rekogClient.send(new
 RecognizeCelebritiesCommand(params));
        console.log(response.Labels)
        response.CelebrityFaces.forEach(celebrity =>{
            console.log(`Name: ${celebrity.Name}`)
            console.log(`ID: ${celebrity.Id}`)
            console.log(`KnownGender: ${celebrity.KnownGender.Type}`)
            console.log(`Smile: ${celebrity.Smile}`)
            console.log('Position: ')
            console.log(`   Left: ${celebrity.Face.BoundingBox.Height}`)
            console.log(`  Top : ${celebrity.Face.BoundingBox.Top}`)

        })
        return response.length; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
}
```

```
recognize_celebrity()
```

.NET

此示例显示与图像中检测到的名人相关的信息。

将photo的值更改为包含一张或多张名人人脸（.jpg 或 .png 格式）的图像文件的路径和文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CelebritiesInImage
{
    public static void Example()
    {
        String photo = "moviestars.jpg";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        RecognizeCelebritiesRequest recognizeCelebritiesRequest = new
 RecognizeCelebritiesRequest();

        Amazon.Rekognition.Model.Image img = new
 Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
            }
        }
```

```
        catch(Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        img.Bytes = new MemoryStream(data);
        recognizeCelebritiesRequest.Image = img;

        Console.WriteLine("Looking for celebrities in image " + photo + "\n");

        RecognizeCelebritiesResponse recognizeCelebritiesResponse =
 rekognitionClient.RecognizeCelebrities(recognizeCelebritiesRequest);

        Console.WriteLine(recognizeCelebritiesResponse.CelebrityFaces.Count + "
 celebrity(s) were recognized.\n");
        foreach (Celebrity celebrity in
 recognizeCelebritiesResponse.CelebrityFaces)
        {
            Console.WriteLine("Celebrity recognized: " + celebrity.Name);
            Console.WriteLine("Celebrity ID: " + celebrity.Id);
            BoundingBox boundingBox = celebrity.Face.BoundingBox;
            Console.WriteLine("position: " +
               boundingBox.Left + " " + boundingBox.Top);
            Console.WriteLine("Further information (if available):");
            foreach (String url in celebrity.Urls)
                Console.WriteLine(url);
        }
        Console.WriteLine(recognizeCelebritiesResponse.UnrecognizedFaces.Count +
 " face(s) were unrecognized.");
    }
}
```

3. 记录显示的其中一位名 IDs 人的价值。您将在获取有关名人的信息中需要它。

RecognizeCelebrities 操作请求

对 RecognizeCelebrities 的输入是一个图像。在此示例中，图像作为图像字节传递。有关更多信息，请参阅 使用图像。

```
{
    "Image": {
        "Bytes": "/AoSiyvFpm....."
```

```
    }
}
```

RecognizeCelebrities 操作响应

下面是 RecognizeCelebrities 的示例 JSON 输入和输出。

RecognizeCelebrities 返回一个已识别名人数组和一个未识别人脸数组。在该示例中，请注意以下内容：

- 已识别的名人 – Celebrities 是已识别名人的数组。数组中的每个名人对象都包含名人姓名和 URLs 指向相关内容的列表，例如名人的IMDB或Wikidata链接。Amazon Rekognition 会ComparedFace返回一个对象，您的应用程序可以使用该对象来确定名人脸部在图片上的位置以及名人的唯一标识符。之后通过 GetCelebrityInfo API 操作使用唯一标识符检索名人信息。
- 未识别的人脸 – UnrecognizedFaces 是与任何已知名人都不匹配的人脸的数组。该数组中的每个 ComparedFace 对象都包含一个您可用于找到人脸在图像上的位置的边界框 (以及其他信息)。

```
{
    "CelebrityFaces": [{
        "Face": {
            "BoundingBox": {
                "Height": 0.617123007774353,
                "Left": 0.15641026198863983,
                "Top": 0.10864841192960739,
                "Width": 0.3641025722026825
            },
            "Confidence": 99.99589538574219,
            "Emotions": [{
                "Confidence": 96.3981749057023,
                "Type": "Happy"
                }
            ],
            "Landmarks": [{
                "Type": "eyeLeft",
                "X": 0.2837241291999817,
                "Y": 0.3637104034423828
            }, {
                "Type": "eyeRight",
                "X": 0.4091649055480957,
                "Y": 0.37378931045532227
```

```
            }, {
                "Type": "nose",
                "X": 0.35267341136932373,
                "Y": 0.49657556414604187
            }, {
                "Type": "mouthLeft",
                "X": 0.2786353826522827,
                "Y": 0.5455248355865479
            }, {
                "Type": "mouthRight",
                "X": 0.39566439390182495,
                "Y": 0.5597742199897766
            }],
            "Pose": {
                "Pitch": -7.749263763427734,
                "Roll": 2.004552125930786,
                "Yaw": 9.012002944946289
            },
            "Quality": {
                "Brightness": 32.69192123413086,
                "Sharpness": 99.9305191040039
            },
            "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
            }
        },
        "Id": "3Ir0du6",
        "KnownGender": {
            "Type": "Male"
        },
        "MatchConfidence": 98.0,
        "Name": "Jeff Bezos",
        "Urls": ["www.imdb.com/name/nm1757263"]
    }],
    "OrientationCorrection": "NULL",
    "UnrecognizedFaces": [{
        "BoundingBox": {
            "Height": 0.5345501899719238,
            "Left": 0.48461538553237915,
            "Top": 0.16949152946472168,
            "Width": 0.3153846263885498
        },
        "Confidence": 99.92860412597656,
```

```
        "Landmarks": [{
            "Type": "eyeLeft",
            "X": 0.5863404870033264,
            "Y": 0.36940744519233704
        }, {
            "Type": "eyeRight",
            "X": 0.6999204754829407,
            "Y": 0.3769848346710205
        }, {
            "Type": "nose",
            "X": 0.6349524259567261,
            "Y": 0.4804527163505554
        }, {
            "Type": "mouthLeft",
            "X": 0.5872702598571777,
            "Y": 0.5535582304000854
        }, {
            "Type": "mouthRight",
            "X": 0.6952020525932312,
            "Y": 0.5600858926773071
        }],
        "Pose": {
            "Pitch": -7.386096477508545,
            "Roll": 2.304218292236328,
            "Yaw": -6.175624370574951
        },
        "Quality": {
            "Brightness": 37.16635513305664,
            "Sharpness": 99.9305191040039
        },
        "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
        }
    }]
}
```


识别存储视频中的名人

存储视频中的 Amazon Rekognition Video 名人识别是异步操作。要识别存储视频中的名人，StartCelebrityRecognition请使用开始视频分析。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service 主题。如果视频分析成功，请调用

GetCelebrityRecognition 来获取分析结果。有关启动视频分析和获取结果的详细信息，请参阅调用 Amazon Rekognition Video 操作。

此过程在 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon SQS 队列获取视频分析请求的完成状态）中的代码的基础上进行了扩展。要运行此过程，您需要一个包含一张或多张名人人脸的视频文件。

检测存储在 Amazon S3 存储桶内的视频中的名人 (SDK)

1. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
2. 将以下代码添加到您在步骤 1 中创建的类 `VideoDetect`。

   Java

```
       //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights
 Reserved.
       //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

     //
 Celebrities============================================================================
     private static void StartCelebrityDetection(String bucket, String video)
 throws Exception{

           NotificationChannel channel= new NotificationChannel()
                   .withSNSTopicArn(snsTopicArn)
                   .withRoleArn(roleArn);

          StartCelebrityRecognitionRequest req = new
 StartCelebrityRecognitionRequest()
                .withVideo(new Video()
                      .withS3Object(new S3Object()
                            .withBucket(bucket)
                            .withName(video)))
                .withNotificationChannel(channel);



          StartCelebrityRecognitionResult startCelebrityRecognitionResult =
 rek.startCelebrityRecognition(req);
          startJobId=startCelebrityRecognitionResult.getJobId();

       }
```

```
       private static void GetCelebrityDetectionResults() throws Exception{

          int maxResults=10;
          String paginationToken=null;
          GetCelebrityRecognitionResult celebrityRecognitionResult=null;

          do{
             if (celebrityRecognitionResult !=null){
                paginationToken = celebrityRecognitionResult.getNextToken();
             }
             celebrityRecognitionResult = rek.getCelebrityRecognition(new
GetCelebrityRecognitionRequest()
                   .withJobId(startJobId)
                   .withNextToken(paginationToken)
                   .withSortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                   .withMaxResults(maxResults));


             System.out.println("File info for page");
             VideoMetadata
videoMetaData=celebrityRecognitionResult.getVideoMetadata();

             System.out.println("Format: " + videoMetaData.getFormat());
             System.out.println("Codec: " + videoMetaData.getCodec());
             System.out.println("Duration: " +
videoMetaData.getDurationMillis());
             System.out.println("FrameRate: " + videoMetaData.getFrameRate());

             System.out.println("Job");

             System.out.println("Job status: " +
celebrityRecognitionResult.getJobStatus());


             //Show celebrities
             List<CelebrityRecognition> celebs=
celebrityRecognitionResult.getCelebrities();

             for (CelebrityRecognition celeb: celebs) {
                long seconds=celeb.getTimestamp()/1000;
                System.out.print("Sec: " + Long.toString(seconds) + " ");
                CelebrityDetail details=celeb.getCelebrity();
                System.out.println("Name: " + details.getName());
```

```
                System.out.println("Id: " + details.getId());
                System.out.println();
            }
         } while (celebrityRecognitionResult !=null &&
celebrityRecognitionResult.getNextToken() != null);

       }
```

在函数 `main` 中，将以下行:

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

       if (GetSQSMessageSuccess()==true)
        GetLabelDetectionResults();
```

替换为:

```
       StartCelebrityDetection(amzn-s3-demo-bucket, video);

       if (GetSQSMessageSuccess()==true)
        GetCelebrityDetectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.recognize_video_celebrity.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
```

```
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_celebrity.import]

/**
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

public class RecognizeCelebritiesVideo {

private static String startJobId ="";

public static void main(String[] args) {

   final String usage = "\n" +
       "Usage: " +
       "   <bucket> <video> <topicArn> <roleArn>\n\n" +
       "Where:\n" +
       "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
       "   video - The name of video (for example, people.mp4). \n\n" +
       "   topicArn - The ARN of the Amazon Simple Notification Service (Amazon
 SNS) topic. \n\n" +
       "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

   if (args.length != 4) {
```

```
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];
    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

   NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

   StartCelebrityDetection(rekClient, channel, bucket, video);
   GetCelebrityDetectionResults(rekClient);
   System.out.println("This example is done!");
   rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_celebrity.main]
public static void StartCelebrityDetection(RekognitionClient rekClient,
                                            NotificationChannel channel,
                                            String bucket,
                                            String video){
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartCelebrityRecognitionRequest recognitionRequest =
 StartCelebrityRecognitionRequest.builder()
            .jobTag("Celebrities")
            .notificationChannel(channel)
```

```
            .video(vidOb)
            .build();

        StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
 rekClient.startCelebrityRecognition(recognitionRequest);
        startJobId = startCelebrityRecognitionResult.jobId();

    } catch(RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetCelebrityDetectionResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetCelebrityRecognitionResponse recognitionResponse = null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (recognitionResponse !=null)
                paginationToken = recognitionResponse.nextToken();

            GetCelebrityRecognitionRequest recognitionRequest =
 GetCelebrityRecognitionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                .maxResults(10)
                .build();

            // Wait until the job succeeds
            while (!finished) {
                recognitionResponse =
 rekClient.getCelebrityRecognition(recognitionRequest);
                status = recognitionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
```

```
                Thread.sleep(1000);
            }
            yy++;
        }

        finished = false;

        // Proceed when the job is done - otherwise VideoMetadata is null.
        VideoMetadata videoMetaData=recognitionResponse.videoMetadata();
        System.out.println("Format: " + videoMetaData.format());
        System.out.println("Codec: " + videoMetaData.codec());
        System.out.println("Duration: " + videoMetaData.durationMillis());
        System.out.println("FrameRate: " + videoMetaData.frameRate());
        System.out.println("Job");

        List<CelebrityRecognition> celebs= recognitionResponse.celebrities();
        for (CelebrityRecognition celeb: celebs) {
            long seconds=celeb.timestamp()/1000;
            System.out.print("Sec: " + seconds + " ");
            CelebrityDetail details=celeb.celebrity();
            System.out.println("Name: " + details.name());
            System.out.println("Id: " + details.id());
            System.out.println();
        }

    } while (recognitionResponse.nextToken() != null);

  } catch(RekognitionException | InterruptedException e) {
      System.out.println(e.getMessage());
      System.exit(1);
  }
}
// snippet-end:[rekognition.java2.recognize_video_celebrity.main]
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Celebrities ===============
    def StartCelebrityDetection(self):
```

```
        response=self.rek.start_celebrity_recognition(Video={'S3Object':
{'Bucket': self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetCelebrityDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_celebrity_recognition(JobId=self.startJobId,
                                                MaxResults=maxResults,
                                                NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for celebrityRecognition in response['Celebrities']:
                print('Celebrity: ' +
                    str(celebrityRecognition['Celebrity']['Name']))
                print('Timestamp: ' + str(celebrityRecognition['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函数 `main` 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    analyzer.StartCelebrityDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetCelebrityDetectionResults()
```

Node.JS

在下面的 Node.Js 代码示例中，将amzn-s3-demo-bucket的值替换为包含您的视频的 S3 存储桶的名称，将 videoName 的值替换为视频文件的名称。您还需要将 roleArn 的值替换为与您的 IAM 服务角色关联的 Arn。最后，将region的值替换为与您的账户关联的运营区域的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Import required AWS SDK clients and commands for Node.js
import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
  SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-
sdk/client-sns";
import  { SQSClient } from "@aws-sdk/client-sqs";
import  { SNSClient } from "@aws-sdk/client-sns";
import  { RekognitionClient, StartLabelDetectionCommand,
 GetLabelDetectionCommand,
  StartCelebrityRecognitionCommand, GetCelebrityRecognitionCommand} from "@aws-
sdk/client-rekognition";
import { stdout } from "process";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
// Create SNS service object.
const sqsClient = new SQSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const snsClient = new SNSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const rekClient = new RekognitionClient({region: REGION,
```

```
  credentials: fromIni({profile: profileName,}),
});

// Set bucket and video variables
const bucket = "bucket-name";
const videoName = "video-name";
const roleArn = "role-arn"
var startJobId = ""

var ts = Date.now();
const snsTopicName = "AmazonRekognitionExample" + ts;
const snsTopicParams = {Name: snsTopicName}
const sqsQueueName = "AmazonRekognitionQueue-" + ts;

 // Set the parameters
 const sqsParams = {
  QueueName: sqsQueueName, //SQS_QUEUE_URL
  Attributes: {
    DelaySeconds: "60", // Number of seconds delay.
    MessageRetentionPeriod: "86400", // Number of seconds delay.
  },
};

const createTopicandQueue = async () => {
  try {
    // Create SNS topic
    const topicResponse = await snsClient.send(new
 CreateTopicCommand(snsTopicParams));
    const topicArn = topicResponse.TopicArn
    console.log("Success", topicResponse);
    // Create SQS Queue
    const sqsResponse = await sqsClient.send(new
 CreateQueueCommand(sqsParams));
    console.log("Success", sqsResponse);
    const sqsQueueCommand = await sqsClient.send(new
 GetQueueUrlCommand({QueueName: sqsQueueName}))
    const sqsQueueUrl = sqsQueueCommand.QueueUrl
    const attribsResponse = await sqsClient.send(new
 GetQueueAttributesCommand({QueueUrl: sqsQueueUrl, AttributeNames:
 ['QueueArn']}))
    const attribs = attribsResponse.Attributes
    console.log(attribs)
    const queueArn = attribs.QueueArn
    // subscribe SQS queue to SNS topic
```

```
    const subscribed = await snsClient.send(new SubscribeCommand({TopicArn:
 topicArn, Protocol:'sqs', Endpoint: queueArn}))
    const policy = {
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "MyPolicy",
          Effect: "Allow",
          Principal: {AWS: "*"},
          Action: "SQS:SendMessage",
          Resource: queueArn,
          Condition: {
            ArnEquals: {
              'aws:SourceArn': topicArn
            }
          }
        }
      ]
    };

    const response = sqsClient.send(new SetQueueAttributesCommand({QueueUrl:
 sqsQueueUrl, Attributes: {Policy: JSON.stringify(policy)}}))
    console.log(response)
    console.log(sqsQueueUrl, topicArn)
    return [sqsQueueUrl, topicArn]

  } catch (err) {
    console.log("Error", err);
  }
};

const startCelebrityDetection = async(roleArn, snsTopicArn) =>{
  try {
      //Initiate label detection and update value of startJobId with returned
 Job ID
      const response = await rekClient.send(new
 StartCelebrityRecognitionCommand({Video:{S3Object:{Bucket:bucket,
 Name:videoName}},
          NotificationChannel:{RoleArn: roleArn, SNSTopicArn: snsTopicArn}}))
          startJobId = response.JobId
          console.log(`Start Job ID: ${startJobId}`)
          return startJobId
    } catch (err) {
      console.log("Error", err);
```

```
    }
  };

const getCelebrityRecognitionResults = async(startJobId) =>{
  try {
      //Initiate label detection and update value of startJobId with returned
 Job ID
      var maxResults = 10
      var paginationToken = ''
      var finished = false

      while (finished == false){
          var response = await rekClient.send(new
 GetCelebrityRecognitionCommand({JobId: startJobId, MaxResults: maxResults,
              NextToken: paginationToken}))
          console.log(response.VideoMetadata.Codec)
          console.log(response.VideoMetadata.DurationMillis)
          console.log(response.VideoMetadata.Format)
          console.log(response.VideoMetadata.FrameRate)
          response.Celebrities.forEach(celebrityRecognition => {
              console.log(`Celebrity: ${celebrityRecognition.Celebrity.Name}`)
              console.log(`Timestamp: ${celebrityRecognition.Timestamp}`)
              console.log()
          })
          // Searh for pagination token, if found, set variable to next token
          if (String(response).includes("NextToken")){
              paginationToken = response.NextToken

          }else{
              finished = true
          }
      }
    } catch (err) {
      console.log("Error", err);
    }
  };

// Checks for status of job completion
const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
  try {
    // Set job found and success status to false initially
    var jobFound = false
    var succeeded = false
    var dotLine = 0
```

```
    // while not found, continue to poll for response
    while (jobFound == false){
      var sqsReceivedResponse = await sqsClient.send(new
ReceiveMessageCommand({QueueUrl:sqsQueueUrl,
       MaxNumberOfMessages:'ALL', MaxNumberOfMessages:10}));
      if (sqsReceivedResponse){
        var responseString = JSON.stringify(sqsReceivedResponse)
        if (!responseString.includes('Body')){
          if (dotLine < 40) {
            console.log('.')
            dotLine = dotLine + 1
          }else {
            console.log('')
            dotLine = 0
          };
          stdout.write('', () => {
            console.log('');
          });
          await new Promise(resolve => setTimeout(resolve, 5000));
          continue
        }
      }

      // Once job found, log Job ID and return true if status is succeeded
      for (var message of sqsReceivedResponse.Messages){
        console.log("Retrieved messages:")
        var notification = JSON.parse(message.Body)
        var rekMessage = JSON.parse(notification.Message)
        var messageJobId = rekMessage.JobId
        if (String(rekMessage.JobId).includes(String(startJobId))){
          console.log('Matching job found:')
          console.log(rekMessage.JobId)
          jobFound = true
          console.log(rekMessage.Status)
          if (String(rekMessage.Status).includes(String("SUCCEEDED"))){
            succeeded = true
            console.log("Job processing succeeded.")
            var sqsDeleteMessage = await sqsClient.send(new
DeleteMessageCommand({QueueUrl:sqsQueueUrl,
ReceiptHandle:message.ReceiptHandle}));
          }
        }else{
          console.log("Provided Job ID did not match returned ID.")
```

```
          var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
        }
      }
    }
  return succeeded
  } catch(err) {
    console.log("Error", err);
  }
};

// Start label detection job, sent status notification, check for success
 status
// Retrieve results if status is "SUCEEDED", delete notification queue and
 topic
const runCelebRecognitionAndGetResults = async () => {
  try {
    const sqsAndTopic = await createTopicandQueue();
    //const startLabelDetectionRes = await startLabelDetection(roleArn,
 sqsAndTopic[1]);
    //const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startLabelDetectionRes)
    const startCelebrityDetectionRes = await startCelebrityDetection(roleArn,
 sqsAndTopic[1]);
    const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startCelebrityDetectionRes)
    console.log(getSQSMessageSuccess)
    if (getSQSMessageSuccess){
      console.log("Retrieving results:")
      const results = await
 getCelebrityRecognitionResults(startCelebrityDetectionRes)
    }
    const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
 sqsAndTopic[0]}));
    const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
 sqsAndTopic[1]}));
    console.log("Successfully deleted.")
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
runCelebRecognitionAndGetResults()
```

CLI

运行以下 AWS CLI 命令开始检测视频中的名人。

```
aws rekognition start-celebrity-recognition --video "{"S3Object":
{"Bucket":"amzn-s3-demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
--region region-name --profile profile-name
```

更新以下值：

- 将amzn-s3-demo-bucket和video-name更改为您在步骤 2 中指定的 Amazon S3 存储桶名称和文件名。
- 将region-name更改为您使用的 AWS 区域。
- 将profile-name的值替换为您的开发人员资料的名称。
- 将 topic-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 3 中创建的 Amazon SNS 主题的 ARN。
- 将 role-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 7 中创建的 IAM 服务角色的 ARN。

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。有关示例，请参阅以下内容：

```
aws rekognition start-celebrity-recognition --video "{\"S3Object\":{\"Bucket\":
\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
--notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
--region region-name --profile profile-name
```

运行后续代码示例后，复制返回的 jobID 并将其提供给以下 GetCelebrityRecognition 命令以获取结果，将 job-id-number 替换为您之前收到的 jobID：

```
aws rekognition get-celebrity-recognition --job-id job-id-number --profile
 profile-name
```

> **Note**
>
> 如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

3. 运行该代码。将显示有关在视频中识别的名人的信息。

GetCelebrityRecognition 操作响应

以下是示例 JSON 响应。此响应包含：

- 已识别的名人 – `Celebrities` 是名人和在视频中识别出他们的时间的数组。每当在视频中识别出名人时，都会存在一个 CelebrityRecognition 对象。每个 `CelebrityRecognition` 都包含有关所识别名人的详细信息 (CelebrityDetail) 和在视频中识别出名人的时间 (`Timestamp`)。`Timestamp` 是从视频起点开始计算的，以毫秒为单位。
- CelebrityDetail— 包含有关知名名人的信息。它包括名人姓名 (`Name`)、标识符 (`ID`)、名人的已知性别 (`KnownGender`) 以及 URLs 指向相关内容的列表 (`Urls`)。它还包括Amazon Rekognition Video 对识别准确性的置信度，以及有关名人面孔的细节。FaceDetail如果您之后需要获取相关内容，可将 `ID` 与 getCelebrityInfo 结合使用。
- VideoMetadata— 有关所分析视频的信息。

```
{
    "Celebrities": [
        {
            "Celebrity": {
                "Confidence": 0.699999988079071,
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20555555820465088,
                        "Left": 0.029374999925494194,
                        "Top": 0.22333332896232605,
                        "Width": 0.11562500149011612
                    },
                    "Confidence": 99.89837646484375,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
```

```
                        "X": 0.06857934594154358,
                        "Y": 0.30842265486717224
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.10396526008844376,
                        "Y": 0.300625205039978
                    },
                    {
                        "Type": "nose",
                        "X": 0.0966852456331253,
                        "Y": 0.34081998467445374
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.075217105448246,
                        "Y": 0.3811396062374115
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.10744428634643555,
                        "Y": 0.37407416105270386
                    }
                ],
                "Pose": {
                    "Pitch": -0.9784082174301147,
                    "Roll": -8.808176040649414,
                    "Yaw": 20.28228759765625
                },
                "Quality": {
                    "Brightness": 43.312068939208984,
                    "Sharpness": 99.9305191040039
                }
            },
            "Id": "XXXXXX",
            "KnownGender": {
                "Type": "Female"
            },
            "Name": "Celeb A",
            "Urls": []
        },
        "Timestamp": 367
    },......
],
```

```
    "JobStatus": "SUCCEEDED",
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

获取有关名人的信息

在这些步骤中，您通过使用 getCelebrityInfo API 操作获取名人信息。名人是通过使用之前对 RecognizeCelebrities 的调用返回的名人 ID 标识的。

正在呼叫 GetCelebrityInfo

这些过程需要 Amazon Rekognition 已知名人的名人 ID。使用您在识别图像中的名人中记住的名人 ID。

获取名人信息 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 使用以下示例调用 GetCelebrityInfo 操作。

   Java

   此示例显示有关名人的姓名和信息。

   id替换为中 IDs 显示的名人之一识别图像中的名人。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoResult;


public class CelebrityInfo {

   public static void main(String[] args) {
      String id = "nnnnnnnn";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      GetCelebrityInfoRequest request = new GetCelebrityInfoRequest()
         .withId(id);

      System.out.println("Getting information for celebrity: " + id);

      GetCelebrityInfoResult
 result=rekognitionClient.getCelebrityInfo(request);

      //Display celebrity information
      System.out.println("celebrity name: " + result.getName());
      System.out.println("Further information (if available):");
      for (String url: result.getUrls()){
         System.out.println(url);
      }
   }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CelebrityInfo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <id>

                Where:
                   id - The id value of the celebrity. You can use the
 RecognizeCelebrities example to get the ID value.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String id = args[0];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        getCelebrityInfo(rekClient, id);
        rekClient.close();
    }

    /**
     * Retrieves information about a celebrity identified in an image.
```

```
     *
     * @param rekClient the Amazon Rekognition client used to make the API call
     * @param id the unique identifier of the celebrity
     * @throws RekognitionException if there is an error retrieving the
 celebrity information
     */
    public static void getCelebrityInfo(RekognitionClient rekClient, String id)
 {
        try {
            GetCelebrityInfoRequest info = GetCelebrityInfoRequest.builder()
                    .id(id)
                    .build();

            GetCelebrityInfoResponse response =
 rekClient.getCelebrityInfo(info);
            System.out.println("celebrity name: " + response.name());
            System.out.println("Further information (if available):");
            for (String url : response.urls()) {
                System.out.println(url);
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

AWS CLI

此 AWS CLI 命令显示 `get-celebrity-info` CLI 操作的 JSON 输出。`ID`替换为中 IDs 显示的名人之一识别图像中的名人。将`profile-name`的值替换为您的开发人员资料的名称。

```
aws rekognition get-celebrity-info --id celebrity-id --profile profile-name
```

Python

此示例显示有关名人的姓名和信息。

`id`替换为中 IDs 显示的名人之一识别图像中的名人。将创建 Rekognition 会话的行中的`profile_name`值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def get_celebrity_info(id):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    # Display celebrity info
    print('Getting celebrity info for celebrity: ' + id)

    response = client.get_celebrity_info(Id=id)

    print(response['Name'])
    print('Further information (if available):')
    for url in response['Urls']:
        print(url)

def main():
    id = "celebrity-id"
    celebrity_info = get_celebrity_info(id)

if __name__ == "__main__":
    main()
```

.NET

此示例显示有关名人的姓名和信息。

id替换为中 IDs 显示的名人之一识别图像中的名人。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
public class CelebrityInfo
{
    public static void Example()
    {
        String id = "nnnnnnnn";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        GetCelebrityInfoRequest celebrityInfoRequest = new
 GetCelebrityInfoRequest()
        {
            Id = id
        };

        Console.WriteLine("Getting information for celebrity: " + id);

        GetCelebrityInfoResponse celebrityInfoResponse =
 rekognitionClient.GetCelebrityInfo(celebrityInfoRequest);

        //Display celebrity information
        Console.WriteLine("celebrity name: " + celebrityInfoResponse.Name);
        Console.WriteLine("Further information (if available):");
        foreach (String url in celebrityInfoResponse.Urls)
            Console.WriteLine(url);
    }
}
```

GetCelebrityInfo 操作请求

下面是 GetCelebrityInfo 的示例 JSON 输入和输出。

GetCelebrityInfo 的输入是所需名人的 ID。

```
{
    "Id": "nnnnnnn"
}
```

GetCelebrityInfo 操作响应

GetCelebrityInfo 返回一个链接数组 (Urls)，这些链接指向有关所请求的名人的信息。

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

审核内容

您可以使用 Amazon Rekognition 来检测不当、不必要或冒犯性内容。您可以在社交媒体、广播媒体、广告和电子商务环境中使用 Rekognition APIs 审核来创造更安全的用户体验，为广告商提供品牌安全保证，并遵守当地和全球法规。

如今，许多公司完全依靠人工审核员来审查第三方或用户生成的内容，而其他公司则只是对用户投诉做出回应，删除冒犯性或不当图片、广告或视频。但是，仅靠人工审核员自身无法以出色的质量或速度进行扩展以满足这些需求，这会导致用户体验不佳，实现规模化的成本高昂，甚至失去品牌声誉。通过使用 Rekognition 进行图像和视频审核，人工审核员可以审核更少的内容，通常是内容总量的 1-5%，且由机器学习标记完毕。这使其能够集中精力开展更有价值的活动，而且仅花费一小部分现有成本即可实现全面审核。要建立人工工作团队并执行人工审核任务，您可以使用 Amazon Augmented AI，它已与 Rekognition 集成。

您可以使用自定义审核功能提高审核深度学习模型的准确性。自定义审核功能让您可以通过上传图像并对这些图像添加注释来训练自定义审核适配器。然后，可以向DetectModerationLabels操作提供经过训练的适配器，以增强其在图像上的性能。请参阅使用自定义审核提高准确性了解更多信息。

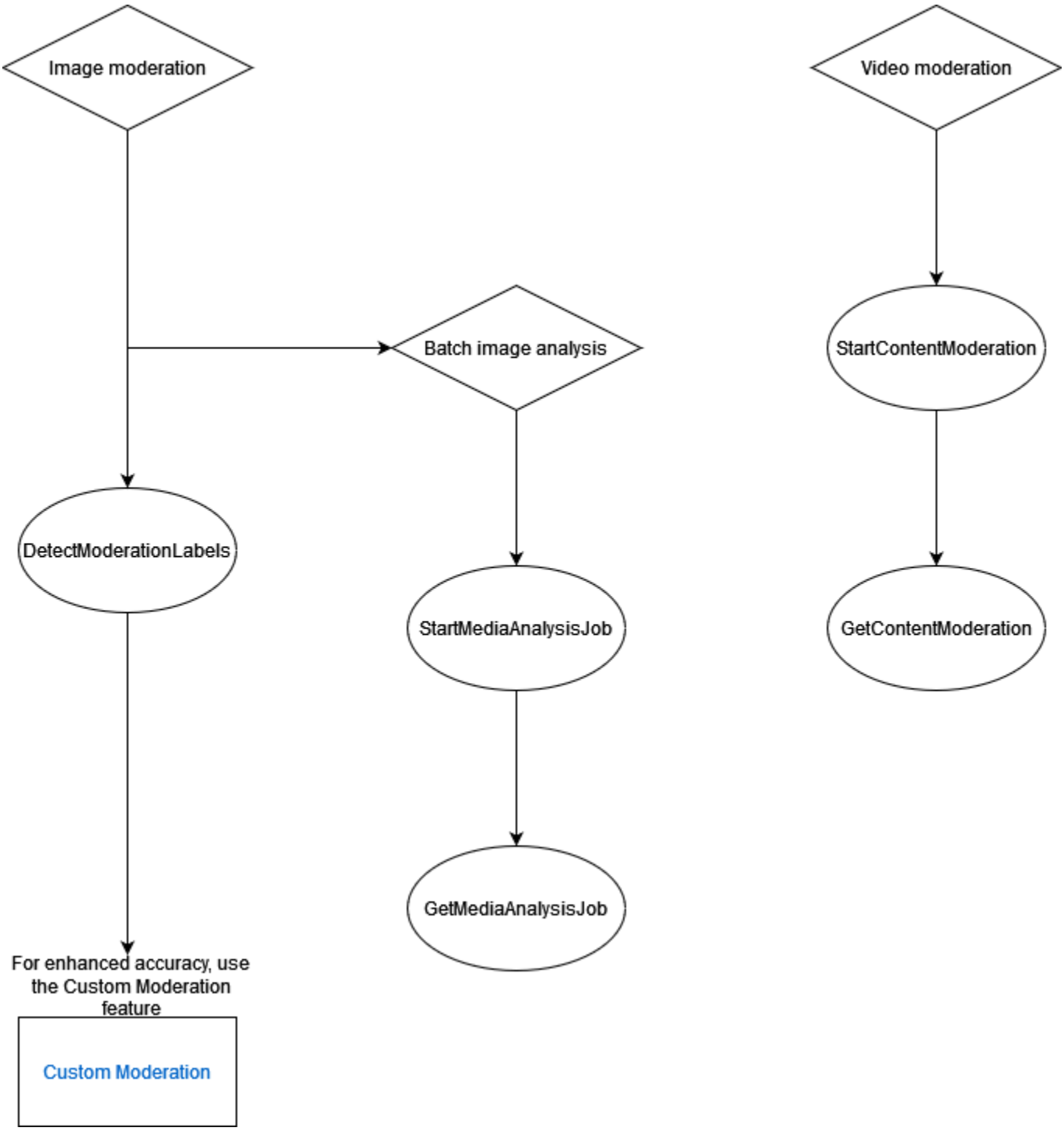
Rekognition 内容审核操作支持的标签

- 要下载审核标签列表，请单击此处。

主题

- 使用图像和视频审核 APIs
- 测试内容审核版本 7 并转换 API 响应
- 检测不当图像
- 检测不当的存储视频
- 使用自定义审核提高准确性
- 使用 Amazon Augmented AI 审核不当内容

下图显示了调用操作的顺序，具体取决于您使用内容审核的图像或视频组件的目标：



使用图像和视频审核 APIs

在 Amazon Rekognition Image API 中，您可以使用和操作以及异步使用和DetectModerationLabels操作同步检测不当、不需要或令人反感的内容。StartMediaAnalysisJobGetMediaAnalysisJob您

可以使用 Amazon Rekognition Video API 通过使用和操作异步检测此类内容。StartContentModerationGetContentModeration

标签类别

Amazon Rekognition 使用三级分层分类来标记不当、不必要或冒犯性内容的类别。每个分类等级 1（L1）的标签都有许多分类等级 2 标签（L2），而某些分类等级 2 标签可能有分类等级 3 标签（L3）。这允许对内容进行分层分类。

对于检测到的每个审核标签，API 还会返回 `TaxonomyLevel`，其中包含该标签所属的等级（1、2 或 3）。例如，可以根据以下分类对一张图像进行标记：

L1：私密部位的不明显裸露以及接吻，L2：明显裸露，L3：暗示裸露。

> **Note**
>
> 我们建议使用 L1 或 L2 类别来审核您的内容，仅使用 L3 类别来移除您不想审核的特定概念（即，根据您的审核政策来检测可能不想归类为不当、不必要或冒犯性内容的内容）。

下表显示了类别级别与每个级别可能的标签之间的关系。要下载审核标签列表，请单击此处。

顶级类别（L1）	第二级类别（L2）	第三级类别（L3）	定义
显式	明显裸露	暴露的男性生殖器	人类男性生殖器，包括阴茎（无论是勃起还是松弛）、阴囊和任何可辨别的阴毛。该术语适用于涉及性行为或任何完全或部分显示男性生殖器的视觉内容的上下文。
		裸露的女性生殖器	女性生殖系统的外部部分，包括外阴、阴道和任何可观察到的阴毛。该术语适用于涉及性活动或任何视觉内容的场景，其中

			女性解剖结构的这些方面全部或部分显示。
		裸露的臀部或肛门	人类的臀部或肛门，包括臀部裸体或通过透明的衣服可以辨别臀部的情况。该定义特别适用于可以直接完全看到臀部或肛门的情况，不包括任何形式的内衣或衣服可以完全或部分覆盖的情况。
		裸露的女性乳头	人类女性乳头，包括完全可见和部分可见的乳头（乳头周围的区域）和乳头。
	露骨的性行为	不适用	描绘实际或模拟的性行为，包括人类性交、口交，以及其他身体部位和物体对男性生殖器刺激和女性生殖器刺激。该术语还包括身体部位的射精或阴道液体，以及涉及束缚、纪律、统治和屈服以及施虐受虐狂的色情行为或角色扮演。
	性玩具	不适用	用于性刺激或愉悦的物体或设备，例如假阳具、振动器、肛塞、节拍等。

私密部位和接吻的非露骨裸体	非露骨裸体	裸露的背部	人体后部，从颈部到脊柱末端可以看到大部分皮肤。当个人的背部被部分或完全遮挡时，该术语不适用。
		裸露的男性乳头	人类男性乳头，包括部分可见的乳头。
		臀部部分暴露	部分暴露的人体臀部。该术语包括由于衣服短而导致的臀部或臀部脸颊部分可见的区域，或者肛门裂缝的顶部部分部分可见。该术语不适用于臀部全裸的情况。
		部分暴露的女性乳房	部分暴露的人类女性乳房，其中女性乳房的一部分可见或未露出，而没有露出整个乳房。该术语适用于乳房内侧褶皱区域可见或乳头完全遮住或遮住乳头时下乳房折痕可见的情况。
		隐含的裸体	一个人是裸体，要么是裸照，要么是露底，但臀部、乳头或生殖器等私密部位被遮住、被遮住或无法完全看见。

	私密部位受阻	女性乳头受阻	视觉描绘了女性的乳头被不透明的衣服或覆盖物遮住，但其形状清晰可见的情况。
		男性生殖器受阻	视觉描绘男性的生殖器或阴茎被不透明的衣服或覆盖物遮住，但其形状清晰可见的情况。该术语适用于图像中受阻的生殖器近距离拍摄的情况。
	亲吻嘴唇	不适用	描绘一个人的嘴唇与另一个人的嘴唇接触。
泳装或内衣	女性泳装或内衣	不适用	女性泳装（例如连体泳衣、比基尼、分体泳衣等）和女性内衣（例如胸罩、内裤、三角裤、内衣、丁字裤等）的人体服装
	男士泳装或内衣	不适用	男士泳装（例如泳裤、沙滩裤、游泳三角裤等）和男士内衣（例如三角裤、平角裤等）的人体服装

暴力	武器	不适用	用于对生物、结构或系统造成伤害或损害的仪器或设备。这包括枪支（例如枪支、步枪、机关枪等）、锋利武器（例如剑、刀等）、爆炸物和弹药（例如导弹、炸弹、子弹等）。
	图形暴力	武器暴力	使用武器对自己、他人或财产造成伤害、损害、伤害或死亡。
		身体暴力	对他人或财产造成伤害的行为（例如打人、打架、拔头发等）或其他涉及人群或多人的暴力行为。
		自我伤害	对自己造成伤害的行为，通常是通过切割手臂或腿部等身体部位，这些部位通常可以看到割伤。
		Blood & Gore	对一个人、一群人或动物施加暴力的视觉表现，涉及开放性伤口、流血事件和身体部位被肢解。
		爆炸和爆炸	描绘了一场猛烈而破坏性的强烈火焰爆发，浓烟或灰尘和烟雾从地面喷出。

视觉干扰	死亡与消瘦	瘦骨嶙峋的身体	人体极度瘦弱，营养不良，身体严重萎缩，肌肉和脂肪组织枯竭。
		尸体	残缺的尸体、悬挂的尸体或骷髅形式的人类尸体。
	崩溃	空难	飞机、直升机或其他飞行器等飞行器导致损坏、受伤或死亡的事故。当飞行器的一部分可见时，该术语适用。
毒品和烟草	产品	药丸	小而坚固，通常是圆形或椭圆形的桌子或胶囊。该术语适用于以独立药丸、瓶装或透明包装形式呈现的药丸，不适用于服用药丸的人的视觉描绘。
	毒品和烟草用具及使用	抽烟	吸入、呼气和点燃燃烧物质的行为，包括香烟、雪茄、电子烟、水烟或关节。
酒精	饮酒	饮酒	用瓶装或杯装酒精或白酒喝含酒精饮料的行为。

	酒精饮料	不适用	近距离观察一瓶或多瓶酒精或烈酒、装有酒精或烈酒的玻璃杯或马克杯，以及个人持有的装有酒精或烈酒的玻璃杯或马克杯。该术语不适用于用瓶装或杯装酒精或酒精饮料的个人。
粗鲁的手势	中指	不适用	用中指向上伸出手势的视觉描绘，而其他手指则向下折叠。
赌博	不适用	不适用	参与机会游戏以有机会在赌场赢取奖品的行为，例如扑克牌、二十一点、轮盘、赌场的老虎机等。
仇恨标志	纳粹党	不适用	与纳粹党相关的符号、旗帜或手势的视觉描绘。
	白人至上主义	不适用	与 Ku Klux Klan（KKK）相关的符号或衣服的视觉描绘以及带有同盟国旗帜的图像。
	极端主义	不适用	包含极端主义和恐怖组织旗帜的图片。

并非 L2 类别中的每个标签都有 L3 类别中支持的标签。此外，“产品”和“毒品和烟草用具及使用”L2 标签下的 L3 标签并非详尽无遗。这些 L2 标签涵盖了提及的 L3 标签之外的概念，在这种情况下，API 响应中仅返回 L2 标签。

您确定内容适合应用程序的程度。例如，可以接受带暗示性内容的图像，但无法接受包含裸体的图像。要筛选图像，请使用 DetectModerationLabels（图像）和 GetContentModeration（视频）返回的 ModerationLabel 标签数组。

内容类型

API 还可以识别动画或插图内容类型，该内容类型将作为响应的一部分返回：

- 动画内容包括视频游戏和动画（例如卡通、连环画、漫画、动画）。
- 插图内容包括素描、绘画和速写。

置信度

您可以通过指定 MinConfidence 输入参数，设置 Amazon Rekognition 在检测不当内容时使用的置信度阈值。不会返回检测的置信度低于 MinConfidence 的不当内容的标签。

指定小于 50% 的 MinConfidence 值时，可能会返回大量假阳性结果（即较高召回率、较低精度）。另一方面，指定大于 50% 的 MinConfidence 可能会返回较少的假阳性结果（即较低召回率、较高精度）。如果您没有为 MinConfidence 指定值，则 Amazon Rekognition 会返回检测的置信度不低于 50% 的不当内容的标签。

ModerationLabel 数组包含前面类别的标签和估计的已识别内容准确度的置信度。顶级标签将与已识别的任何第二级标签一起返回。例如，Amazon Rekognition 可能返回具有高置信度得分的“明显裸露”作为顶级标签。这也许能够满足筛选需求。但如有必要，您可以使用第二级标签（例如“男性裸体图”）的置信度得分来实现更高粒度的筛选。有关示例，请参阅检测不当图像。

版本控制

Amazon Rekognition Image 和 Amazon Rekognition Video 都返回了用于检测不当内容的审核检测模型的版本（ModerationModelVersion）。

排序和汇总

使用检索结果时 GetContentModeration，您可以对结果进行排序和汇总。

排序顺序 – 返回的标签数组按时间进行排序。要按标签进行排序，请为 GetContentModeration 在 SortBy 输入参数中指定 NAME。如果此标签在视频中多次出现，则会有 ModerationLabel 元素的多个实例。

标签信息 — ModerationLabels 数组元素包含一个ModerationLabel对象，该对象又包含标签名称以及 Amazon Rekognition 对检测到的标签的准确性的信心。时间戳是检测到 ModerationLabel 的时间，定义为视频开始后经过的毫秒数。对于按视频SEGMENTS汇总的结果，将返回 StartTimestampMillis、EndTimestampMillis 和 DurationMillis 结构，它们分别定义了片段的开始时间、结束时间和持续时间。

汇总 – 指定返回结果时的汇总方式。默认为按 TIMESTAMPS 汇总。您也可以选择按 SEGMENTS 汇总，即在某个时间段内汇总结果。只返回在分段期间检测到的标签。

自定义审核适配器状态

自定义审核适配器可能处于以下状态之一：TRAINING_IN_PROGRESS、TRAINING_COMPLETED、TRAINING_FAILED、DELETING、DEPRECATED 或 EXPIRED。有关这些适配器状态的完整说明，请参阅管理适配器。

> **Note**
>
> Amazon Rekognition 既不是权威，也不以任何方式声称会彻底筛选不当或冒犯性内容。此外，图片和视频审核 APIs 不会检测图片是否包含非法内容，例如 CSAM。

测试内容审核版本 7 并转换 API 响应

Rekognition 将内容审核标签检测功能中图像视频组件的机器学习模型从版本 6.1 更新到版本 7。此更新提高了整体准确性，引入了几个新类别并修改了其他类别。

如果您当前是版本 6.1 的视频用户，我们建议您采取以下操作无缝过渡到版本 7：

1. 下载并使用 AWS 私有软件开发工具包（参见the section called “AWS 内容审核 SDK 和使用指南版本 7”）来调用 StartContentModeration API。
2. 查看 API 响应或控制台中返回的标签和置信度分数的更新列表。如有必要，相应地调整应用程序的后处理逻辑。
3. 您的账户可在 2024 年 5 月 13 日之前使用版本 6.1。如果您希望在 2024 年 5 月 13 日之后使用版本 6.1，请在 2024 年 4 月 30 日之前与 AWS Support 团队联系，申请延期。我们可以为您延期，让您的账户在 2024 年 6 月 10 日之前继续使用版本 6.1。如果我们在 2024 年 4 月 30 日之前没有收到您的回复，则您的账户将从 2024 年 5 月 13 日起自动迁移到版本 7.0。

AWS 内容审核 SDK 和使用指南版本 7

下载与您选择的开发语言相对应的 SDK，并查阅相应的用户指南。

指向 SDK 的链接	安装/用户指南
Java-1.X	指南——Java 1.pdf
Java-2.X	指南——Java 2.pdf
JavaScript v2	指南- JavaScript v2.pdf
JavaScript v3	指南- JavaScript v3.pdf
Python	指南——Python 和 AWS CLI.pdf
Ruby	指南-RubyV3.pdf
go_v1	指南——GO V1.pdf
go_v2	指南——GO V2.pdf
DotNet	指南-.net.pdf
php	指南-PHP.pdf

版本 6.1 到版本 7 的标签映射

内容审核版本 7 添加了新的标签类别并修改了之前存在的标签名称。在决定如何将 6.1 标签映射到 7 标签时，请参考the section called “ 标签类别 ”中的分类表。

下一节提供了一些示例标签映射。建议您在根据应用程序的后处理逻辑进行必要的更新之前，先查看这些映射和标签定义。

L1 映射架构

如果您使用仅对顶级类别（L1）（例如 `Explicit Nudity`、`Suggestive`、`Violence` 等）进行筛选的后处理逻辑，请参阅下表来更新您的代码。

V6.1 L1	V7 L1

明显裸露	显式
暗示性	私密部位和接吻的非露骨裸体
	泳装或内衣
暴力	暴力
视觉干扰	视觉干扰
粗鲁的手势	粗鲁的手势
毒品	毒品和烟草
烟草	毒品和烟草
酒精	酒精
赌博	赌博
仇恨标志	仇恨标志

L2 映射架构

如果您使用对 L1 和 L2 类别（例如 `Explicit Nudity / Nudity, Suggestive / Female Swimwear Or Underwear`、`Violence / Weapon Violence` 等）进行筛选的后处理逻辑，请参阅下表来更新您的代码。

V6.1 L1	V6.1 L2	V7 L1	V7 L2	V7 L3	V7 ContentTypes
明显裸露	裸露	显式	明显裸露	裸露的女性乳头	
				裸露的臀部或肛门	
	男性裸体图	显式	明显裸露	暴露的男性生殖器	

	女性裸体图	显式	明显裸露	裸露的女性生殖器	
	性行为	显式	露骨的性行为		
	明显裸露插图	显式	明显裸露		映射到 “动画” 和 “插图”
	明显裸露插图	显式	露骨的性行为		映射到 “动画” 和 “插图”
	成人玩具	显式	性玩具		
暗示性	女性泳衣或内衣	泳装或内衣	女性泳装或内衣		
	男性泳衣或内衣	泳装或内衣	男士泳装或内衣		
	部分裸露	私密部位和接吻的非露骨裸体	非露骨裸体	隐含的裸体	
	赤膊男性	私密部位和接吻的非露骨裸体	非露骨裸体	裸露的男性乳头	
	暴露的衣服	私密部位和接吻的非露骨裸体	非露骨裸体		
		私密部位和接吻的非露骨裸体	私密部位受阻		
	色情场景	私密部位和接吻的非露骨裸体	亲吻嘴唇		

暴力	暴力或血腥画面	暴力	图形暴力	Blood & Gore
	身体暴力	暴力	图形暴力	身体暴力
	武器暴力	暴力	图形暴力	武器暴力
	武器	暴力	武器	
	自我伤害	暴力	图形暴力	自我伤害
视觉干扰	瘦骨嶙峋的身体	视觉干扰	死亡与消瘦	瘦骨嶙峋的身体
	尸体	视觉干扰	死亡与消瘦	尸体
	自缢	视觉干扰	死亡与消瘦	尸体
	空难	视觉干扰	崩溃	空难
	爆炸	暴力	图形暴力	爆炸和爆炸
粗鲁的手势	中指	粗鲁的手势	中指	
毒品	药品	毒品和烟草	产品	
	吸毒	毒品和烟草	毒品和烟草用具及使用	
	药丸	毒品和烟草	产品	药丸
	吸毒用具	毒品和烟草	毒品和烟草用具及使用	
烟草	烟草制品	毒品和烟草	产品	
	抽烟	毒品和烟草	毒品和烟草用具及使用	抽烟
酒精	饮酒	酒精	饮酒	饮酒
	酒精饮料	酒精	酒精饮料	

赌博	赌博	赌博	
仇恨标志	纳粹党	仇恨标志	纳粹党
	白人至上主义	仇恨标志	白人至上主义
	极端主义	仇恨标志	极端主义

检测不当图像

您可以使用该DetectModerationLabels操作来确定图片是否包含不当内容或令人反感的内容。有关亚马逊 Rekognition 中的审核标签列表，请参阅使用图片和视频审核。 APIs

检测图像中的不当内容

图像的格式必须为 .jpg 或 .png。您可以提供输入图像作为图像字节数组（base64 编码的图像字节）或指定 Amazon S3 对象。在这些过程中，请将图像（.jpg 或 .png）上传到 S3 存储桶中。

要运行这些过程，您需要安装 AWS CLI 或相应的 AWS SDK。有关更多信息，请参阅 Amazon Rekognition 入门。您使用的 AWS 账户必须具有访问 Amazon Rekognition API 的权限。有关更多信息，请参阅 Amazon Rekognition 定义的操作。

检测图像中的审阅标签 (SDK)

1. 如果您尚未执行以下操作，请：

   a. 使用 `AmazonRekognitionFullAccess` 和 `AmazonS3ReadOnlyAccess` 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将图像上传到 S3 存储桶。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 `DetectModerationLabels` 操作。

Java

此示例输出检测到的不当内容标签名称、置信度级别以及检测到的审核标签的父标签。

将 amzn-s3-demo-bucket 和 photo 的值替换为您在步骤 2 中使用的 S3 存储桶名称和图像文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectModerationLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectModerationLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ModerationLabel;
import com.amazonaws.services.rekognition.model.S3Object;

import java.util.List;

public class DetectModerationLabels
{
   public static void main(String[] args) throws Exception
   {
      String photo = "input.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      DetectModerationLabelsRequest request = new
 DetectModerationLabelsRequest()
        .withImage(new Image().withS3Object(new
 S3Object().withName(photo).withBucket(bucket)))
        .withMinConfidence(60F);
      try
      {
           DetectModerationLabelsResult result =
 rekognitionClient.detectModerationLabels(request);
```

```
            List<ModerationLabel> labels = result.getModerationLabels();
            System.out.println("Detected labels for " + photo);
            for (ModerationLabel label : labels)
            {
               System.out.println("Label: " + label.getName()
                + "\n Confidence: " + label.getConfidence().toString() + "%"
                + "\n Parent:" + label.getParentName());
            }
        }
        catch (AmazonRekognitionException e)
        {
          e.printStackTrace();
        }
    }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
//snippet-start:[rekognition.java2.detect_mod_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.ModerationLabel;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_mod_labels.import]

/**
```

```
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ModerateLabels {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length < 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     detectModLabels(rekClient, sourceImage);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.detect_mod_labels.main]
 public static void detectModLabels(RekognitionClient rekClient, String
 sourceImage) {

     try {
         InputStream sourceStream = new FileInputStream(sourceImage);
         SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
         Image souImage = Image.builder()
             .bytes(sourceBytes)
```

```
            .build();

        DetectModerationLabelsRequest moderationLabelsRequest =
DetectModerationLabelsRequest.builder()
            .image(souImage)
            .minConfidence(60F)
            .build();

        DetectModerationLabelsResponse moderationLabelsResponse =
rekClient.detectModerationLabels(moderationLabelsRequest);
        List<ModerationLabel> labels =
moderationLabelsResponse.moderationLabels();
        System.out.println("Detected labels for image");

        for (ModerationLabel label : labels) {
            System.out.println("Label: " + label.name()
                + "\n Confidence: " + label.confidence().toString() + "%"
                + "\n Parent:" + label.parentName());
        }

    } catch (RekognitionException | FileNotFoundException e) {
        e.printStackTrace();
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.detect_mod_labels.main]
```

AWS CLI

此 AWS CLI 命令显示 `detect-moderation-labels` CLI 操作的 JSON 输出。

将 `amzn-s3-demo-bucket` 和 `input.jpg` 替换为您在步骤 2 中使用的 S3 存储桶名称和图像文件名称。将`profile_name`的值替换为您的开发人员资料的名称。要使用适配器，请为`project-version`参数提供项目版本的 ARN。

```
aws rekognition detect-moderation-labels --image "{S3Object:{Bucket:<amzn-s3-
demo-bucket>,Name:<image-name>}}" \
--profile profile-name \
--project-version "ARN"
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition detect-moderation-labels --image "{\"S3Object\":{\"Bucket\":
\"amzn-s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--profile profile-name
```

Python

此示例输出检测到的不当或冒犯性内容标签名称、置信度级别以及检测到的不当内容标签的父标签。

在函数 main 中，将 amzn-s3-demo-bucket 和 photo 的值替换为您在步骤 2 中使用的 S3 存储桶名称和图像文件名。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

def moderate_image(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')

    response = client.detect_moderation_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}})

    print('Detected labels for ' + photo)
    for label in response['ModerationLabels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))
        print (label['ParentName'])
    return len(response['ModerationLabels'])

def main():

    photo='image-name'
    bucket='amzn-s3-demo-bucket'
    label_count=moderate_image(photo, bucket)
    print("Labels detected: " + str(label_count))

if __name__ == "__main__":
```

```
    main()
```

.NET

此示例输出检测到的不当或冒犯性内容标签名称、置信度级别以及检测到的审核标签的父标签。

将 amzn-s3-demo-bucket 和 photo 的值替换为您在步骤 2 中使用的 S3 存储桶名称和图像文件名。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectModerationLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectModerationLabelsRequest detectModerationLabelsRequest = new
 DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MinConfidence = 60F
        };

        try
```

```
        {
            DetectModerationLabelsResponse detectModerationLabelsResponse =
 rekognitionClient.DetectModerationLabels(detectModerationLabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (ModerationLabel label in
 detectModerationLabelsResponse.ModerationLabels)
                Console.WriteLine("Label: {0}\n Confidence: {1}\n Parent: {2}",
                    label.Name, label.Confidence, label.ParentName);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectModerationLabels 操作请求

对 DetectModerationLabels 的输入是一个图像。在此示例 JSON 输入中，源图像从 Amazon S3 存储桶加载。MinConfidence 是 Amazon Rekognition Image 对检测到的标签要在响应中返回而对其准确度所具有的最小置信度。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "MinConfidence": 60
}
```

DetectModerationLabels 操作响应

DetectModerationLabels 可以从 S3 存储桶检索输入图像，也可通过图像字节形式提供输入图像。以下示例是来自对 DetectModerationLabels 的调用的响应。

在以下示例 JSON 响应中，注意以下几点：

- 不当图像检测信息 – 该示例显示了图像中发现的不当或冒犯性内容的标签列表。此列表包括在图像中检测到的顶级标签和所有第二级标签。

标签 – 每个标签具有一个名称、Amazon Rekognition 估计的置信度（用于指示标签的准确性）以及其父标签的名称。顶级标签的父名称为 ""。

标签置信度 – 每个标签均有一个介于 0 和 100 之间的置信度值，该值指示 Amazon Rekognition 具有的百分比置信度（用于指示标签的准确性）。您可以在 API 操作请求中指定要在响应中返回的标签的所需置信度级别。


```
{
    "ModerationLabels": [
        {
            "Confidence": 99.44782257080078,
            "Name": "Smoking",
            "ParentName": "Drugs & Tobacco Paraphernalia & Use",
            "TaxonomyLevel": 3
        },
        {
            "Confidence": 99.44782257080078,
            "Name": "Drugs & Tobacco Paraphernalia & Use",
            "ParentName": "Drugs & Tobacco",
            "TaxonomyLevel": 2
        },
        {
            "Confidence": 99.44782257080078,
            "Name": "Drugs & Tobacco",
            "ParentName": "",
            "TaxonomyLevel": 1
        }
    ],
    "ModerationModelVersion": "7.0",
    "ContentTypes": [
        {
            "Confidence": 99.9999008178711,
            "Name": "Illustrated"
        }
    ]
}
```

检测不当的存储视频

Amazon Rekognition Video 在存储视频中检测不当或冒犯性内容是一种异步操作。要开始检测不当或令人反感的内容，请致电StartContentModeration。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service 主题。如果视频分析成功，请调用 GetContentModeration 来获取分析结果。有关启动视频分析和获取结果的详细信息，请参阅调用 Amazon Rekognition Video 操作。有关亚马逊 Rekognition 中的审核标签列表，请参阅使用图片和视频审核。 APIs

此过程在使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon Simple Queue Service 队列获取视频分析请求的完成状态）中的代码的基础上进行了扩展。

检测存储在 Amazon S3 存储桶 (SDK) 的视频中的不当或冒犯性内容

1. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
2. 将以下代码添加到您在步骤 1 中创建的类 `VideoDetect`。

   Java

```
    //Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
    //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

        //Content moderation
 ======================================================================
        private static void StartUnsafeContentDetection(String bucket, String
 video) throws Exception{

            NotificationChannel channel= new NotificationChannel()
                    .withSNSTopicArn(snsTopicArn)
                    .withRoleArn(roleArn);

            StartContentModerationRequest req = new
 StartContentModerationRequest()
                    .withVideo(new Video()
                            .withS3Object(new S3Object()
                                .withBucket(bucket)
                                .withName(video)))
                    .withNotificationChannel(channel);
```

```
            StartContentModerationResult startModerationLabelDetectionResult =
rek.startContentModeration(req);
            startJobId=startModerationLabelDetectionResult.getJobId();

        }

        private static void GetUnsafeContentDetectionResults() throws
Exception{

            int maxResults=10;
            String paginationToken=null;
            GetContentModerationResult moderationLabelDetectionResult =null;

            do{
                if (moderationLabelDetectionResult !=null){
                    paginationToken =
moderationLabelDetectionResult.getNextToken();
                }

                moderationLabelDetectionResult = rek.getContentModeration(
                        new GetContentModerationRequest()
                            .withJobId(startJobId)
                            .withNextToken(paginationToken)
                            .withSortBy(ContentModerationSortBy.TIMESTAMP)
                            .withMaxResults(maxResults));



                VideoMetadata
videoMetaData=moderationLabelDetectionResult.getVideoMetadata();

                System.out.println("Format: " + videoMetaData.getFormat());
                System.out.println("Codec: " + videoMetaData.getCodec());
                System.out.println("Duration: " +
videoMetaData.getDurationMillis());
                System.out.println("FrameRate: " +
videoMetaData.getFrameRate());


                //Show moderated content labels, confidence and detection
times
                List<ContentModerationDetection> moderationLabelsInFrames=
                        moderationLabelDetectionResult.getModerationLabels();
```

```
                for (ContentModerationDetection label:
 moderationLabelsInFrames) {
                    long seconds=label.getTimestamp()/1000;
                    System.out.print("Sec: " + Long.toString(seconds));
                    System.out.println(label.getModerationLabel().toString());
                    System.out.println();
                }
            } while (moderationLabelDetectionResult !=null &&
 moderationLabelDetectionResult.getNextToken() != null);

        }
```

在函数 `main` 中，将以下行:

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

替换为:

```
        StartUnsafeContentDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetUnsafeContentDetectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
```

```
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
```

```
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startModerationDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {

        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                    .jobTag("Moderation")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartContentModerationResponse startModDetectionResult = rekClient
                    .startContentModeration(modDetectionRequest);
            startJobId = startModDetectionResult.jobId();
```

```
        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getModResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetContentModerationResponse modDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (modDetectionResponse != null)
                    paginationToken = modDetectionResponse.nextToken();

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
```

```
                VideoMetadata videoMetaData =
 modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<ContentModerationDetection> mods =
 modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod : mods) {
                    long seconds = mod.timestamp() / 1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }

            } while (modDetectionResponse != null &&
 modDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    # ============== Unsafe content ===============
    def StartUnsafeContent(self):
        response=self.rek.start_content_moderation(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)
```

```
    def GetUnsafeContentResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_content_moderation(JobId=self.startJobId,
                                                       MaxResults=maxResults,
                                                       NextToken=paginationToken,
                                                       SortBy="NAME",
                                                       AggregateBy="TIMESTAMPS")

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for contentModerationDetection in response['ModerationLabels']:
                print('Label: ' +
                    str(contentModerationDetection['ModerationLabel']['Name']))
                print('Confidence: ' +
                    str(contentModerationDetection['ModerationLabel']
['Confidence']))
                print('Parent category: ' +
                    str(contentModerationDetection['ModerationLabel']
['ParentName']))
                print('Timestamp: ' +
 str(contentModerationDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函数 `main` 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    analyzer.StartUnsafeContent()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetUnsafeContentResults()
```

> **Note**
>
> 如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

3. 运行该代码。将显示在视频中检测到的不当内容标签的列表。

GetContentModeration 操作响应

GetContentModeration 的响应为数组 ModerationLabels，其中包括 ContentModerationDetection 对象。每当检测到不当内容标签时，该数组就包含一个元素。在ContentModerationDetectionObject对象中，ModerationLabel包含检测到的不当或令人反感内容的项目的信息。 Timestamp是检测到标签的时间，以视频开头以毫秒为单位。标签将按不当内容图像分析检测标签的同一方式分层组织。有关更多信息，请参阅 审核内容。

以下是来自 GetContentModeration 的示例响应，按NAME排序，按TIMESTAMPS汇总。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
```

```
                "Name": "Alcohol",
                "ParentName": "",
                "TaxonomyLevel": 1
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        },
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcoholic Beverages",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "TIMESTAMPS"
    }
}
```

以下是来自 GetContentModeration 的示例响应，按NAME排序，按SEGMENTS汇总。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 0,
            "ModerationLabel": {
                "Confidence": 0.0003000000142492354,
                "Name": "Alcohol Use",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 29520,
            "DurationMillis": 29520,
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Illustrated"
                },
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
```

```
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "SEGMENTS"
    }
}
```

使用自定义审核提高准确性

Amazon Rekognition 的 DetectModerationLabelsAPI 允许您检测不当、不需要或令人反感的内容。Rekognition 自定义审核功能允许您使用适配器来提高准确性。DetectModerationLabels适配器是模块化组件，可以添加到现有 Rekognition 深度学习模型中，从而扩展其用于训练任务的功能。通过创建适配器并将其提供给DetectModerationLabels操作，您可以提高与您的特定用例相关的内容审核任务的准确性。

为特定的审核标签自定义 Rekognition 的内容审核模型时，您必须创建一个项目，并在您提供的一组图像上训练适配器。然后，您可以反复检查适配器的性能，并将适配器重新训练到所需的准确性水平。项目用于包含不同版本的适配器。

您可以使用 Rekognition 控制台创建项目和适配器。或者，您可以使用 S AWS DK 和关联的 APIs 来创建项目、训练适配器并管理您的适配器。

创建和使用适配器

适配器是模块化组件，可以添加到现有 Rekognition 深度学习模型中，从而扩展其用于训练任务的功能。通过使用适配器训练深度学习模型，您可以提高与特定用例相关的图像分析任务的准确性。

要创建和使用适配器，您必须向 Rekognition 提供训练和测试数据。您可以通过以下两种方式之一实现这一点：

- 批量分析和验证 – 您可以通过批量分析 Rekognition 将分析并分配标签的图像来创建训练数据集。然后，您可以查看为图像生成的注释并验证或更正预测。有关图像批量分析工作原理的更多信息，请参阅批量分析。
- 手动注释 – 使用这种方法，您可以通过上传图像并对其进行注释来创建训练数据。您可以通过上传图像并对其进行注释或自动拆分来创建测试数据。

请选择下列主题之一以了解更多信息：

主题

- 批量分析和验证
- 手动注释

批量分析和验证

使用这种方法，您可以上传大量要用作训练数据的图像，然后使用 Rekognition 来预测这些图像，从而自动为它们分配标签。您可以使用这些预测开始构建适配器。您可以验证预测的准确性，然后根据经过验证的预测来训练适配器。这可以通过 AWS 控制台来完成。

批量分析和自定义审核

上传图像以进行批量分析

要为您的适配器创建训练数据集，请批量上传图像，让 Rekognition 预测其标签。为获得最佳结果，请尽可能多地提供用于训练的图像，上限不超过 10000 张，并确保这些图像能代表您的用例的各个方面。

使用 AWS 控制台时，您可以直接从计算机上传图片，也可以提供用于存储图像的 Amazon 简单存储服务存储桶。但是，将 APIs Rekognition 与软件开发工具包配合使用时，必须提供一个清单文件，该文件引用存储在亚马逊简单存储服务存储桶中的图像。有关更多信息，请参阅批量分析。

查看预测

将图像上传到 Rekognition 控制台后，Rekognition 将为其生成标签。然后，您可以根据以下几类结果验证预测：真阳性、假阳性、真阴性、假阴性。验证预测后，您可以根据反馈训练适配器。

训练适配器

验证完批量分析返回的预测后，即可启动适配器的训练过程。

去拿吧 AdapterId

适配器经过训练后，您可以获得适配器的唯一 ID，用于 Rekognition 的图像分析。 APIs

调用 API 操作

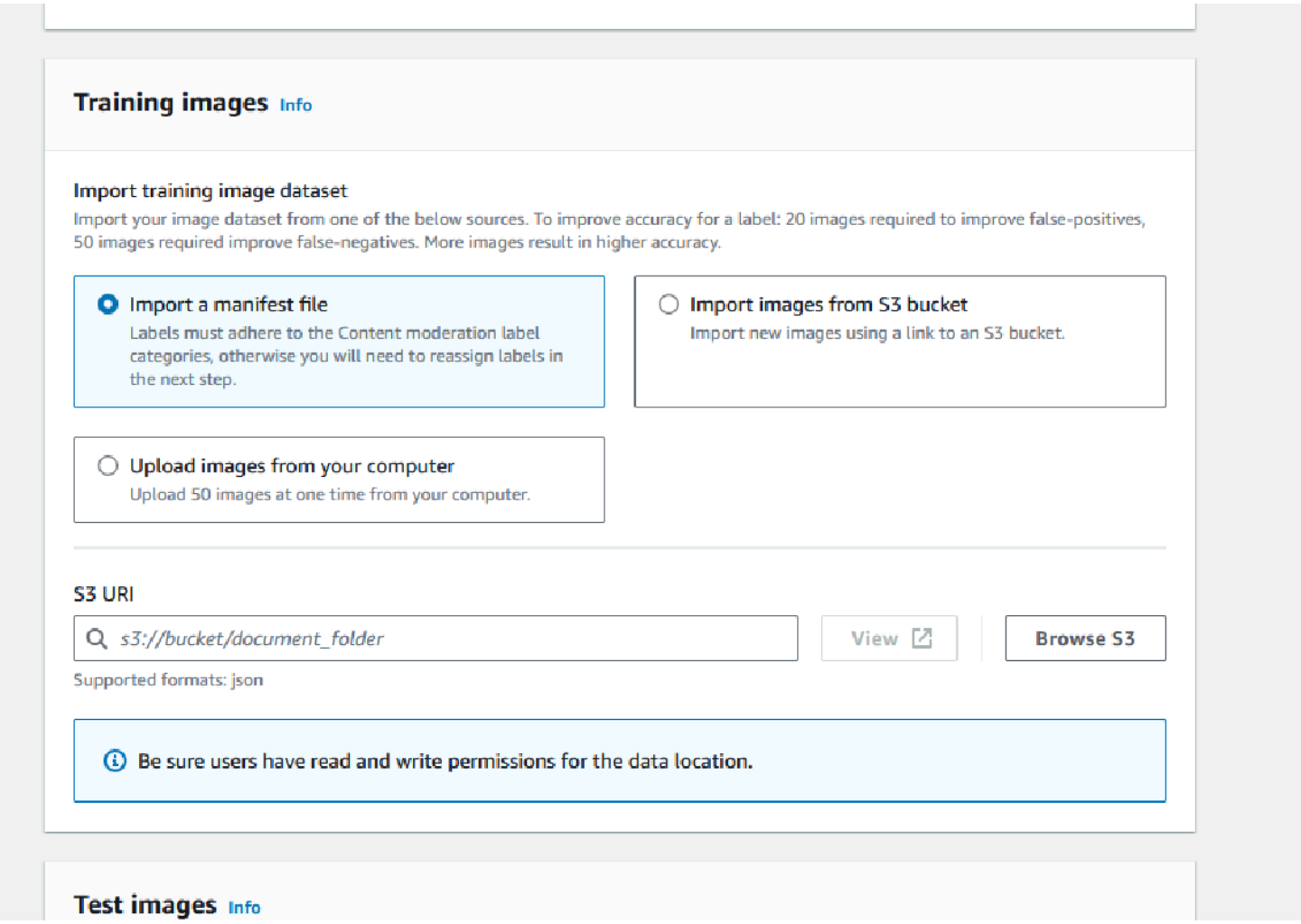
要应用您的自定义适配器，请在调用支持适配器的图像分析 APIs 时提供其 ID。这可以提高图像预测的准确性。

手动注释

使用这种方法，您可以通过手动上传图像并对其进行注释来创建训练数据。您可以通过上传图像并对其进行注释或自动拆分来创建测试数据，从而让 Rekognition 自动使用部分训练数据作为测试图像。

上传图像并对其进行注释

要训练适配器，您需要上传一组代表您的用例的示例图像。为获得最佳结果，请尽可能多地提供用于训练的图像，上限不超过 10000 张，并确保这些图像能代表您的用例的各个方面。



使用 AWS 控制台时，您可以直接从计算机上传图像、提供清单文件或提供用于存储图像的 Amazon S3 存储桶。

但是，将 APIs Rekognition 与软件开发工具包配合使用时，必须提供一个引用存储在 Amazon S3 存储桶中的图像的清单文件。

您可以使用 Rekognition 控制台的注释界面为图像添加注释。通过用标签标记图像来对其进行注释，这样可以为训练建立一个“真实情况”。在训练适配器之前，您还必须指定训练集和测试集，或者使用自动拆分功能。指定数据集并对图像进行注释后，您可以根据测试集中的带注释的图像创建适配器。然后，您可以评估适配器的性能。

创建测试集

您需要提供带注释的测试集或使用自动拆分功能。训练集用于实际训练适配器。适配器学习这些带注释的图像中包含的图案。测试集用于在最终确定适配器之前评估模型的性能。

训练适配器

完成对训练数据的注释或提供了清单文件后，即可启动适配器的训练过程。

获取适配器 ID

适配器经过训练后，您可以获得适配器的唯一 ID，用于 Rekognition 的图像分析。 APIs

调用 API 操作

要应用您的自定义适配器，请在调用支持适配器的图像分析 APIs 时提供其 ID。这可以提高图像预测的准确性。

准备数据集

创建适配器需要您为 Rekognition 提供两个数据集，即训练数据集和测试数据集。每个数据集都包含两个元素：图像和注释/标签。以下各节说明了标签和图像的用途，以及如何将它们组合在一起创建数据集。

映像

您需要在有代表性的图像样本上训练适配器。在选择用于训练的图像时，请尽量包含至少几张图像，以演示适配器所针对的每个标签的预期响应。

要创建训练数据集，您需要提供以下两种图像类型之一：

- 预测为假阳性的图像。例如，当基本模型预测图像中有酒精，但实际却没有。
- 预测为假阴性的图像。例如，当基本模型预测图像中没有酒精，但实际却有。

要创建平衡的数据集，建议您提供以下两种图像类型之一：

- 预测为真阳性的图像。例如，当基本模型正确预测图像中含有酒精时。如果您提供假阳性图像，建议您提供这些图像。
- 预测为真阴性的图像。例如，当基本模型正确预测图像中不含酒精时。如果您提供假阴性图像，建议您提供这些图像。

标签

标签是指以下任何内容：对象、事件、概念或活动。对于内容审核，标签是指不当、不必要或冒犯性内容的实例。

在通过训练 Rekognition 的基础模型来创建适配器的过程中，为图像分配一个标签就叫做注释。使用 Rekognition 控制台训练适配器时，您将使用该控制台通过选择标签然后标记与标签对应的图像来为图像添加注释。通过此过程，模型学会根据指定的标签识别图像中的元素。这种链接过程可以让模型在创建适配器时将重点放在最相关的内容上，从而提高图像分析的准确性。

或者，您可以提供清单文件，其中包含有关图像以及与之相关的注释的信息。

训练和测试数据集

训练数据集是微调模型和创建自定义适配器的基础。您必须提供带注释的训练数据集供模型学习。模型会从该数据集中学习，以提高其在处理您提供的图像类型时的性能。

为提高准确性，您必须对图像进行注释/标记，以创建训练数据集。您可以通过两种方式实现这一点：

- 手动分配标签 – 您可以使用 Rekognition 控制台创建训练数据集，方法是上传您希望数据集包含的图像，然后手动为这些图像分配标签。
- 清单文件 – 您可以使用清单文件来训练适配器。清单文件包含有关训练和测试图像的真实情况注释以及训练图像的位置信息。在使用 APIs Rekognition 训练适配器或使用控制台时，您可以提供清单文件。 AWS

测试数据集用于在训练后评估适配器的性能。为确保评估的可靠性，测试数据集是通过使用模型以前从未见过的原始训练数据集片段来创建的。此过程可确保使用新数据评测适配器的性能，从而建立准确的衡量标准和衡量尺度。要获得最佳精度改进，请参阅训练适配器的最佳实践。

使用 AWS CLI 管理适配器和 SDKs

Rekognition 可让您使用利用预先训练的计算机视觉模型的多种功能。使用这些模型，您可以执行标签检测和内容审核等任务。您也可以使用适配器自定义这些特定型号。

你可以利用 Rekognition 的项目创建和项目管理 APIs（比如CreateProject和 CreateProjectVersion）来创建和训练适配器。以下页面介绍如何使用 AWS 控制台、您选择的 AWS SDK 或 CLI 使用 AP AWS I 操作创建、训练和管理适配器。

训练适配器后，可以在使用支持的功能运行推理时使用它。当前，使用内容审核功能时支持适配器。

使用 AWS SDK 训练适配器时，必须以清单文件的形式提供真实情况标签（图像注释）。或者，您可以使用 Rekognition 控制器创建和训练适配器。

> **Note**
>
> 无法复制适配器。只能复制 Rekognition 自定义标签项目版本。

主题

- 适配器状态
- 创建项目
- 描述项目
- 删除项目
- 创建项目版本
- 描述项目版本
- 删除项目版本

适配器状态

自定义审核适配器（项目版本）可以处于下列状态之一：

- TRAINING_IN_PROGRESS：适配器正在根据您提供作为培训文档的文件进行训练。
- TRAINING_COMPLETED：适配器已成功完成训练，您可以随时查看其性能。
- TRAINING_FAILED：适配器由于某种原因而未能完成训练，请查看输出清单文件和输出清单摘要，了解有关失败原因的信息。
- DELETING：正在删除适配器。
- DEPRECATED：适配器已在旧版本的内容审核基础模型上进行训练。它处于宽限期，将在新的基础模型版本发布后的 60 到 90 天内过期。在宽限期内，您仍然可以使用适配器通过DetectModerationLabels或 StartMediaAnalysisJobAPI 操作进行推理。有关适配器的到期日期，请参阅自定义审核控制台。

- 已过期-适配器已在旧版本的内容审核基础模型上进行训练，无法再用于通过 DetectModerationLabels 或 StartMediaAnalysisJob API 操作获取自定义结果。如果在推理请求中指定了过期适配器，则将忽略该适配器，而是从最新版自定义审核基础模型返回响应。

创建项目

通过该CreateProject操作，您可以创建一个项目，该项目将包含用于 Rekognition 标签检测操作的适配器。项目是一组资源，就标签检测操作而言，项目允许您存储适配器 DetectModerationLabels，您可以使用这些适配器来自定义基本 Rekognition 模型。调用时 CreateProject，您需要为 ProjectName 参数提供要创建的项目的名称。

要使用 AWS 控制台创建项目，请执行以下操作：

- 登录 Rekognition 控制台
- 单击自定义审核
- 选择创建项目
- 选择创建新项目或添加到现有项目
- 添加项目名称
- 添加适配器名称
- 如果需要，可以添加描述
- 选择要导入训练图像的方式：通过清单文件导入、从 S3 存储桶或计算机导入
- 选择是要自动拆分训练数据还是要导入清单文件
- 选择是否要让项目自动更新
- 单击创建项目

要使用 AWS CLI 和 SDK 创建项目，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码创建项目:

CLI

```
# Request
```

```
# Creating Content Moderation Project
aws rekognition create-project \
    --project-name "project-name" \
    --feature CONTENT_MODERATION \
    --auto-update ENABLED
    --profile profile-name
```

描述项目

您可以使用 DescribeProjectsAPI 获取有关您的项目的信息，包括与项目关联的所有适配器的信息。

要使用 AWS CLI 和 SDK 描述项目，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码描述项目：

CLI

```
# Request
# Getting CONTENT_MODERATION project details
aws rekognition describe-projects \
    --features CONTENT_MODERATION
    --profile profile-name
```

删除项目

您可以使用 Rekognition 控制台或调用 API 来删除项目。DeleteProject要删除项目，必须先删除所有关联的适配器。已删除的项目或模型无法取消删除。

要使用 AWS 控制台删除项目，请执行以下操作：

- 登录 Rekognition 控制台。
- 单击自定义审核。
- 您必须先删除与项目关联的所有适配器，然后才能删除项目本身。选择适配器，然后选择删除，即可删除与项目关联的所有适配器。

- 选择项目，然后选择删除按钮。

要使用 AWS CLI 和 SDK 删除项目，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码删除项目：

CLI

```
aws rekognition delete-project
  --project-arn project_arn \
  --profile profile-name
```

创建项目版本

您可以使用CreateProjectVersion操作训练适配器进行部署。 CreateProjectVersion 首先创建与项目关联的适配器的新版本，然后开始训练该适配器。来自的响应 CreateProjectVersion 是模型版本的亚马逊资源名称 (ARN)。训练需要一段时间才能完成。您可以通过致电获取当前状态 DescribeProjectVersions。训练模型时，Rekognition 使用与项目关联的训练和测试数据集。使用控制台创建数据集。有关更多信息，请参阅数据集上的部分。

要使用 Rekognition 控制台创建项目版本，请执行以下操作：

- 登录 AWS Rekognition 控制台
- 单击自定义审核
- 选择项目。
- 在“项目详细信息”页面上，选择创建适配器
- 在“创建项目”页面上，填写项目详细信息、训练图像和测试图像的必填详细信息，然后选择创建项目。
- 在“为图像分配标签”页面上，为图像添加标签，完成后选择开始训练

要使用 AWS CLI 和 SDK 创建项目版本，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码创建项目版本：

CLI

```
# Request
aws rekognition create-project-version \
 --project-arn project-arn \
 --training-data '{Assets=[GroundTruthManifest={S3Object="amzn-s3-demo-source-bucket",Name="manifest.json"}]}' \
 --output-config S3Bucket=amzn-s3-demo-destination-bucket,S3KeyPrefix=my-results \
 --feature-config "ContentModeration={ConfidenceThreshold=70}"
 --profile profile-name
```

描述项目版本

您可以使用DescribeProjectVersions操作列出和描述与项目关联的适配器。您最多可以在中指定 10 个模型版本 ProjectVersionArns。如果不指定值，则返回项目中所有模型版本的描述。

要使用 AWS CLI 和 SDK 描述项目版本，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码来描述项目版本：

CLI

```
aws rekognition describe-project-versions
  --project-arn project_arn \
  --version-names [versions]
```

删除项目版本

您可以使用该操作删除与项目关联的 Rekognition 适配器。DeleteProjectVersion无法删除正在运行或正在训练的适配器。要检查适配器的状态，请调用该 DescribeProjectVersions 操作并检查其返回的“状态” 字段。要停止正在运行的适配器，请调用 StopProjectVersion。如果模型正在训练，请等到训练完成后再删除模型。您必须先删除与项目关联的所有适配器，然后才能删除项目本身。

要使用 Rekognition 控制台删除项目版本，请执行以下操作：

- 登录 Rekognition 控制台
- 单击自定义审核
- 在“项目”选项卡中，您可以查看所有项目和关联的适配器。选择适配器，然后选择删除。

要使用 AWS CLI 和 SDK 删除项目版本，请执行以下操作：

1. 如果您尚未这样做，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 使用以下代码删除项目版本：

CLI

```
# Request
aws rekognition delete-project-version
  --project-version-arn model_arn \
  --profile profile-name
```

自定义审核适配器教程

本教程向您展示如何使用 Rekognition 控制台创建、训练、评估、使用和管理适配器。要使用 AWS SDK 创建、使用和管理适配器，请参阅使用 AWS CLI 管理适配器和 SDKs。

通过适配器，您可以提高 Rekognition API 操作的准确性，自定义模型的行为，以满足您自己的需求和用例。使用本教程创建适配器后，您将能够在分析自己的图像时使用它 DetectModerationLabels，并通过诸如此类的操作以及重新训练适配器以进行进一步的改进。

在本教程中，您将学习如何：

- 使用 Rekognition 控制器创建项目
- 对您的训练数据进行注释
- 在训练数据集上训练您的适配器
- 查看您的适配器的性能
- 使用您的适配器分析图像

先决条件

在完成本教程之前，建议您通读一遍创建和使用适配器。

要创建适配器，您可以使用 Rekognition 控制器工具创建项目，上传自己的图像并添加注释，然后在这些图像上训练适配器。要开始，请参阅创建项目并训练适配器。

或者，您可以使用 Rekognition 控制台或 API 来检索图像的预测，然后在根据这些预测训练适配器之前验证预测。要开始，请参阅批量分析、预测验证和训练适配器。

图像注释

您可以使用 Rekognition 控制台对图像进行标记，或者使用 Rekognition 批量分析对图像进行注释，然后验证标记是否正确。从以下主题中选择一个主题开始。

主题

- 创建项目并训练适配器
- 批量分析、预测验证和训练适配器

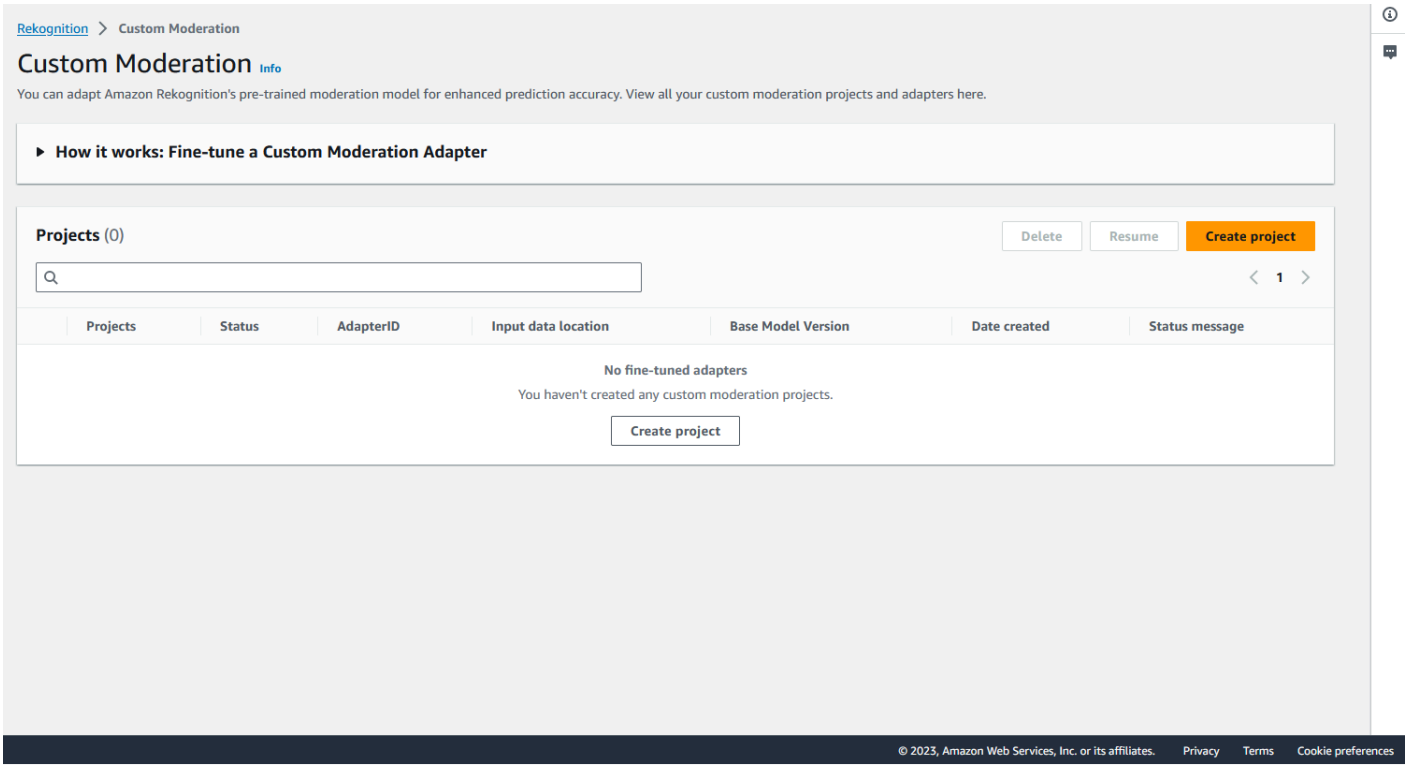
创建项目并训练适配器

完成以下步骤，通过使用 Rekognition 控制台为图像添加注释来训练适配器。

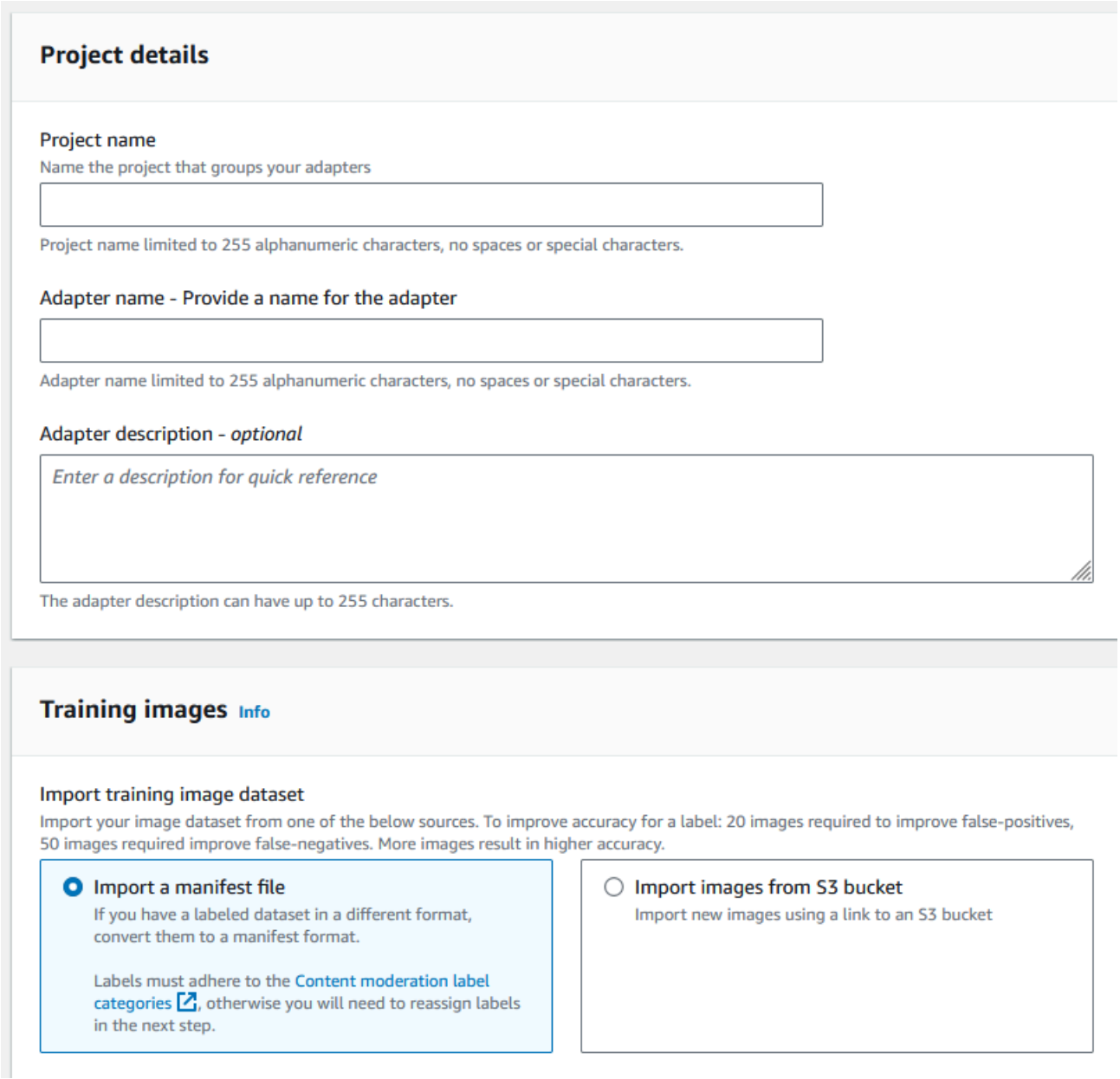
创建项目

在训练或使用适配器之前，必须先创建包含该适配器的项目。您还必须提供用于训练适配器的图像。要创建项目、适配器和图像数据集，请执行以下操作：

1. 登录 AWS 管理控制台并打开 Rekognition 控制台，网址为。 https://console.aws.amazon.com/rekognition/
2. 在左窗格中，选择自定义审核。将显示 Rekognition 自定义审核登录页面。



3. 自定义审核登录页面显示了所有项目和适配器的列表，还有一个用于创建适配器的按钮。选择创建项目创建新项目和适配器。
4. 如果这是您首次创建适配器，系统将提示您创建 Amazon S3 存储桶来存储与您的项目和适配器相关的文件。选择创建 Amazon S3 存储桶。
5. 在下一页上，填写适配器名称和项目名称。如果需要，请提供适配器说明。

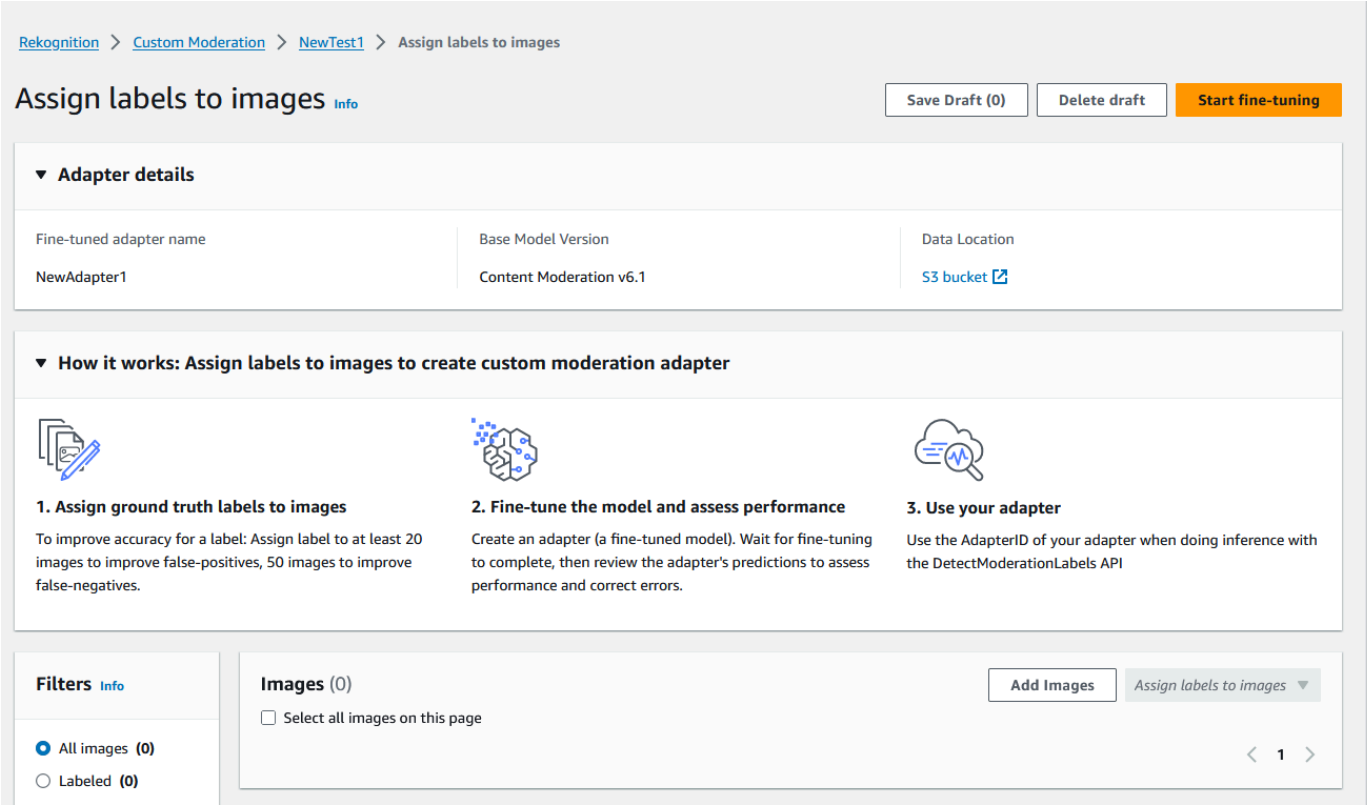


6. 在此步骤中，您还将为适配器提供图像。您可以选择：从计算机导入图像、导入清单文件或从 Amazon S3 存储桶导入图像。如果您选择从 Amazon S3 存储桶导入图像，请提供存储桶的路径以及包含您的训练图像的文件夹。如果您直接从计算机上传图片，请注意，一次最多只能上传 30 张图片。如果使用的清单文件包含注释，则可以跳过下面列出的图像注释步骤，直接进入 查看适配器性能 部分。
7. 在测试数据集详细信息部分，选择自动拆分，让 Rekognition 自动选择相应比例的图像作为测试数据，或者您可以选择手动导入清单文件。
8. 填写此信息后，选择创建项目。

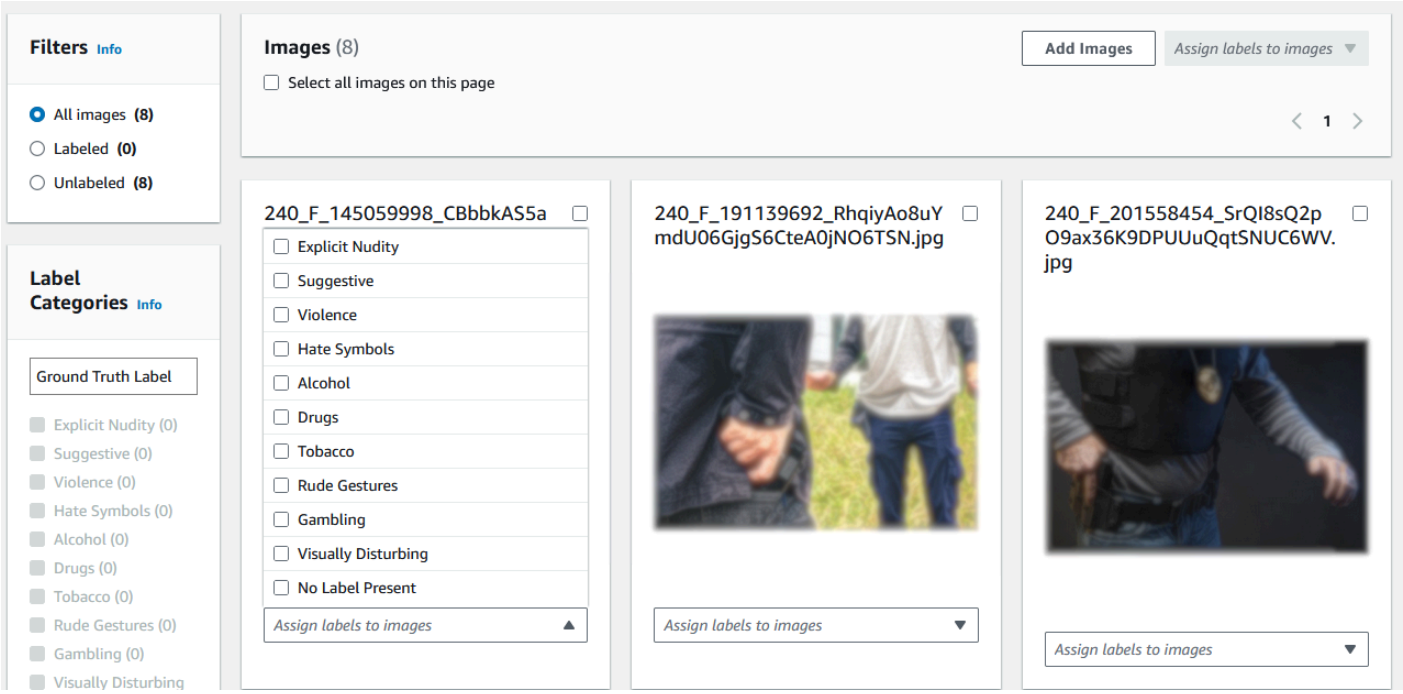
训练适配器

要在您自己的未加注释的图像上训练适配器，请执行以下操作：

1. 选择包含您的适配器的项目，然后选择为图像分配标签选项。
2. 在为图像分配标签页面上，您可以看到所有已作为训练图像上传的图像。您可以使用左侧的两个属性选择面板按已标记/未标记状态和标签类别筛选这些图像。您可以通过选择添加图像按钮将其他图像添加到训练数据集中。



3. 将图像添加到训练数据集后，必须使用标签为图像添加注释。上传图像后，“为图像分配标签”页面将更新以显示您上传的图像。系统将提示您从 Rekognition Moderation 支持的标签下拉列表中选择适合您图像的标签。您可以选择多个标签。
4. 继续此过程，直到为训练数据中的所有图像添加标签为止。
5. 标记完所有数据后，选择开始训练以开始训练模型，从而创建适配器。



6. 在开始训练过程之前，您可以根据需要向适配器添加任何标签。您也可以为适配器提供自定义加密密钥或使用 AWS KMS 密钥。添加完所需标签并根据自己的喜好自定义加密后，选择训练适配器开始适配器的训练过程。
7. 等待您的适配器完成训练。训练完成后，您将收到一条通知，告知您的适配器已完成创建。

适配器的状态变为“训练已完成”后，您可以查看适配器的指标

批量分析、预测验证和训练适配器

通过验证 Rekognition 内容审核模型中的批量分析预测，完成以下步骤来训练您的适配器。

要通过验证 Rekognition 内容审核模型的预测来训练适配器，您必须：

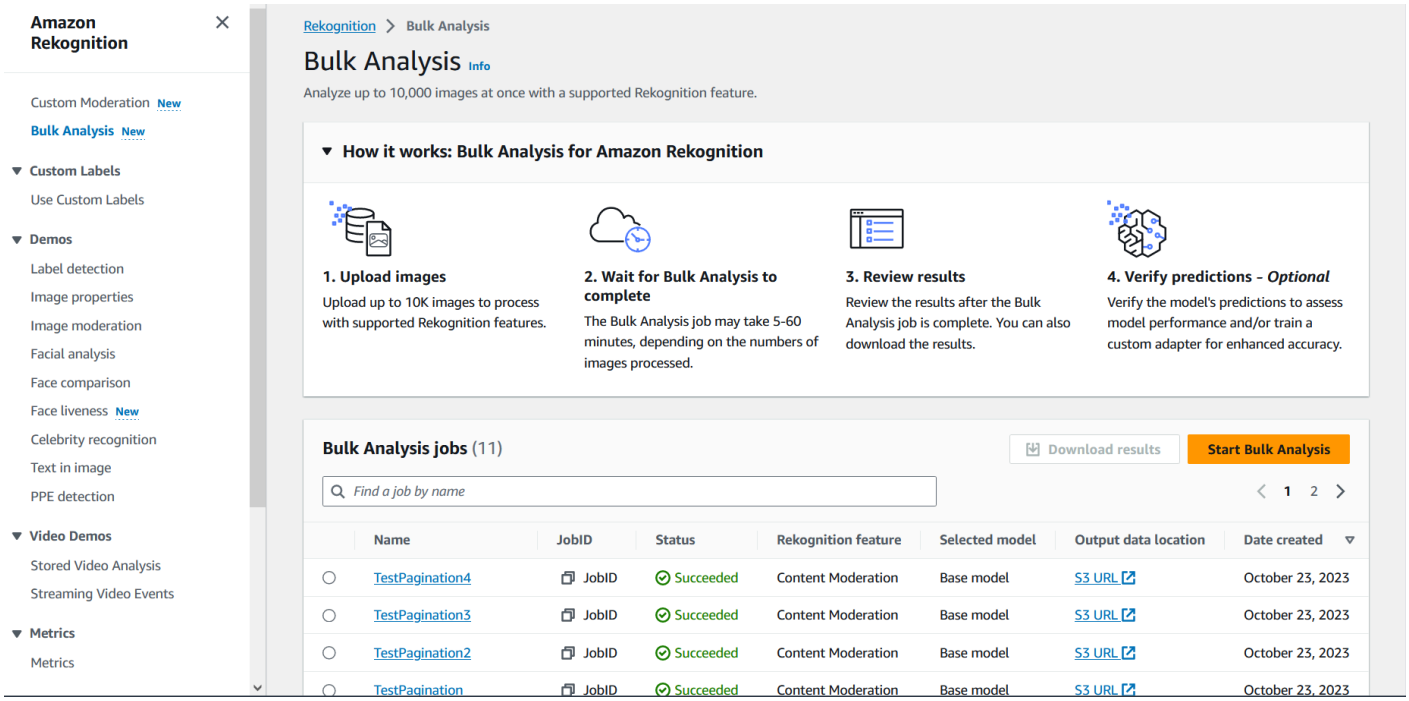
1. 对您的图像进行批量分析
2. 验证返回的图像预测结果

您可以使用 Rekognition 的基础模型或已创建的适配器进行批量分析，从而获得对图像的预测。

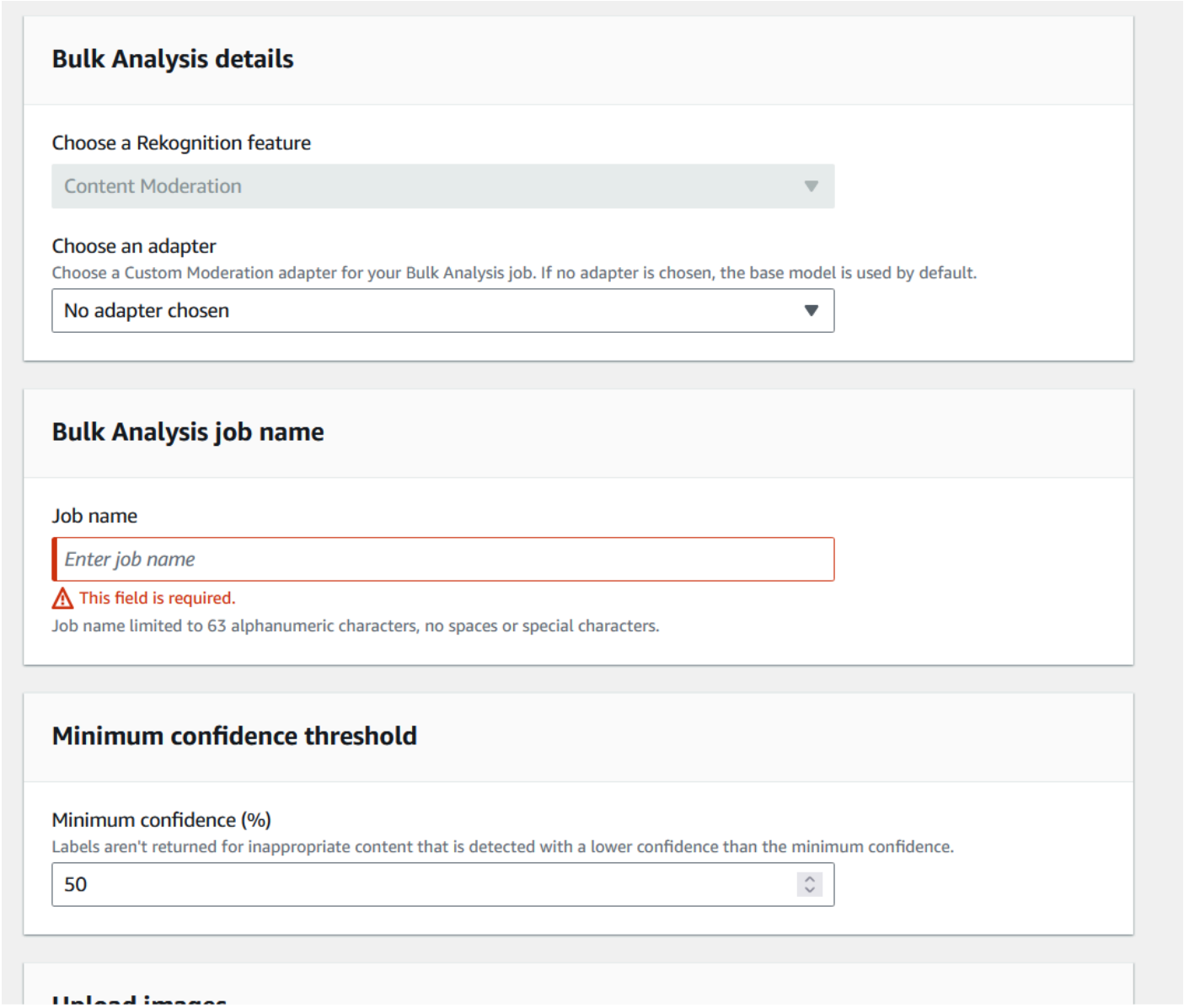
对您的图像运行批量分析

要根据已验证的预测对适配器进行训练，必须首先启动批量分析任务，使用 Rekognition 的基本模型或您选择的适配器分析一批图像。要运行批量分析任务，请执行以下操作：

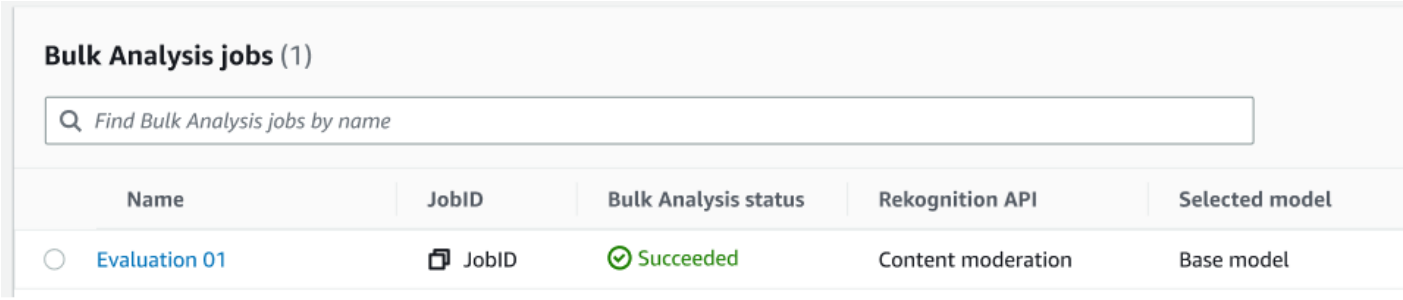
1. 登录 AWS Management Console 并打开亚马逊 Rekognition 控制台，网址为。https://console.aws.amazon.com/rekognition/
2. 在左窗格中，选择批量分析。批量分析登录页面随即出现。选择开始批量分析。批量分析特征概述中显示了上传图像、等待分析、查看结果以及可选地验证模型预测等步骤。列出内容审核近期使用基础模型执行的批量分析作业。



3. 如果这是您首次创建适配器，系统将提示您创建 Amazon Simple Storage Service 存储桶来存储与您的项目和适配器相关的文件。选择创建 Amazon S3 存储桶。
4. 使用选择适配器下拉菜单选择要用于批量分析的适配器。如果未选择适配器，则默认使用基础模型。在本教程中，请勿选择适配器。



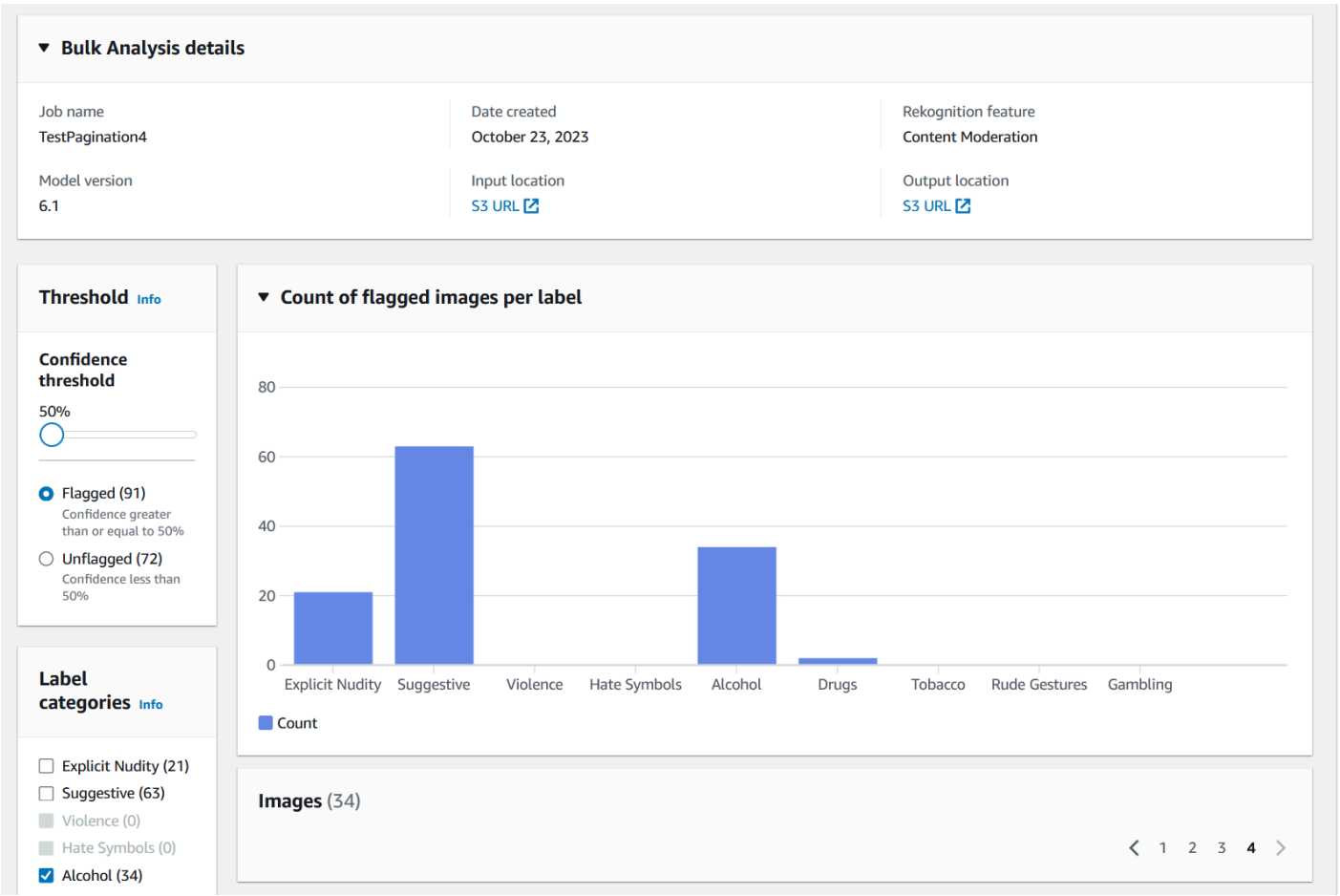
5. 在批量分析任务名称字段中，填写批量分析任务名称。
6. 为最低置信度阈值选择一个值。如果标签预测值小于您选择的置信度阈值，则不会返回。请注意，在以后评估模型性能时，您将无法把置信度阈值调整到低于您选择的最小置信度阈值。
7. 在此步骤中，您还将提供要通过批量分析进行分析的图像。这些图像也可用于训练您的适配器。您可以选择从计算机上传图像或从 Amazon S3 存储桶导入图像。如果您选择从 Amazon S3 存储桶导入文档，请提供存储桶的路径以及包含您的训练图像的文件夹。如果您直接从计算机上传文档，请注意，一次只能上传 50 张图片。
8. 填写此信息后，选择开始分析。这将使用 Rekognition 的基础模型开始分析过程。
9. 您可以通过在批量分析主页面上检查任务的批量分析状态来检查批量分析任务的状态。当批量分析状态变为“成功”时，分析结果已准备就绪，可供查看。



10从批量分析作业列表中选择您创建的分析。

11.在批量分析详情页面上，您可以看到 Rekognition 的基础模型对您上传的图像所做的预测。

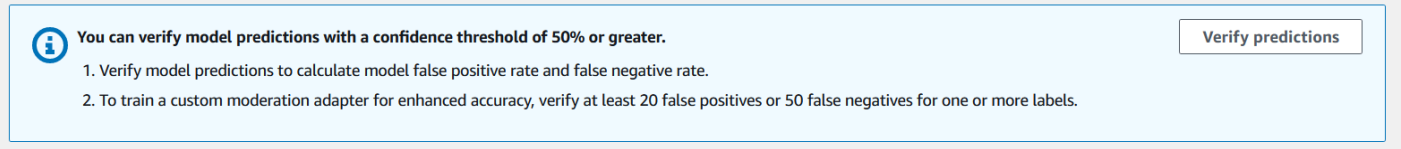
12查看基础模型的性能。您可以使用置信度阈值滑块来更改您的适配器为图像分配标签时必须达到的置信度阈值。已标记和未标记实例的数量会随着置信度阈值的调整而变化。标签类别面板显示 Rekognition 识别的顶级类别，您可以在此列表中选择一个类别来显示已分配该标签的所有图像。



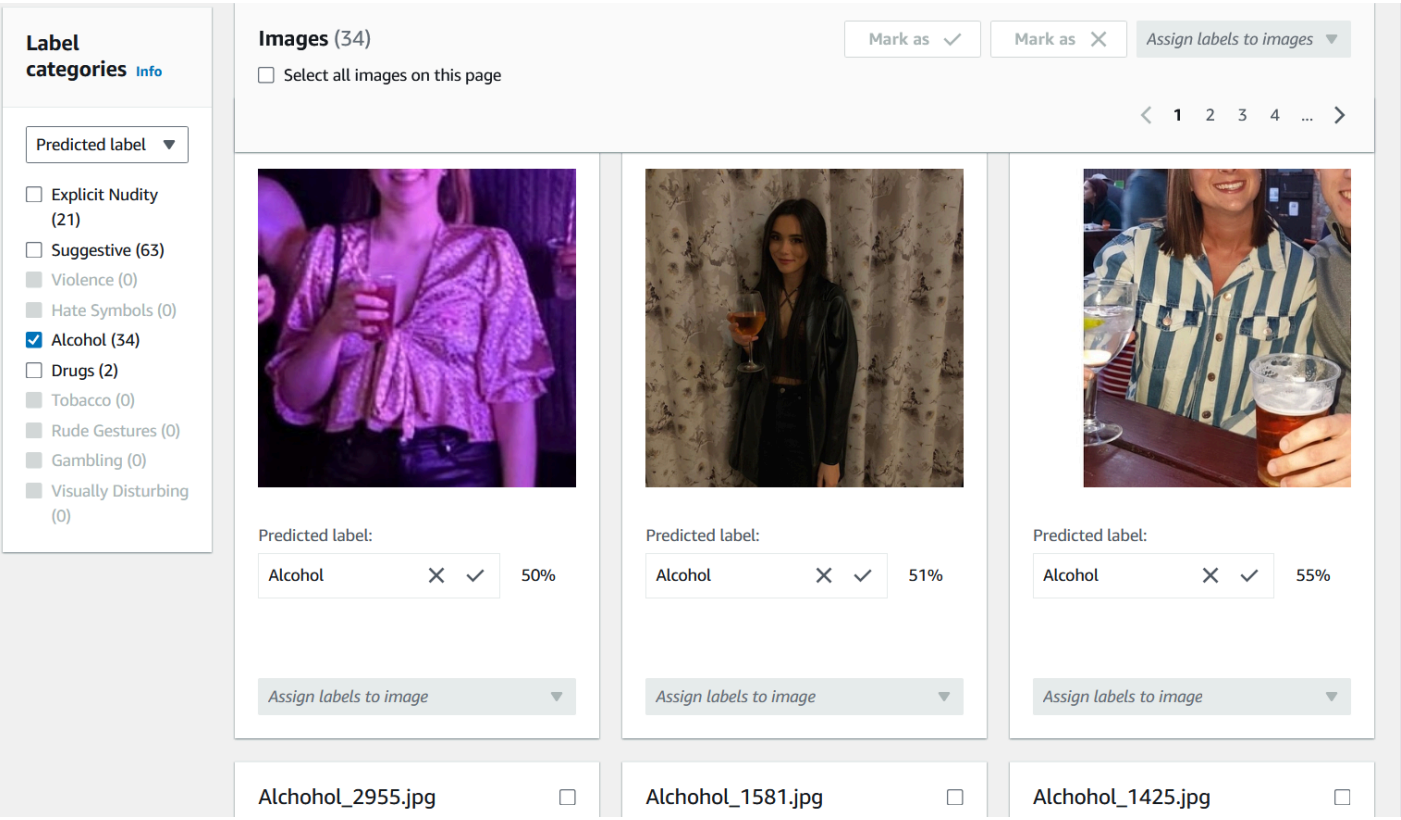
验证预测

如果您已经查看了 Rekognition 的基本模型或所选适配器的准确性，并且想要提高准确性，则可以使用验证工作流程：

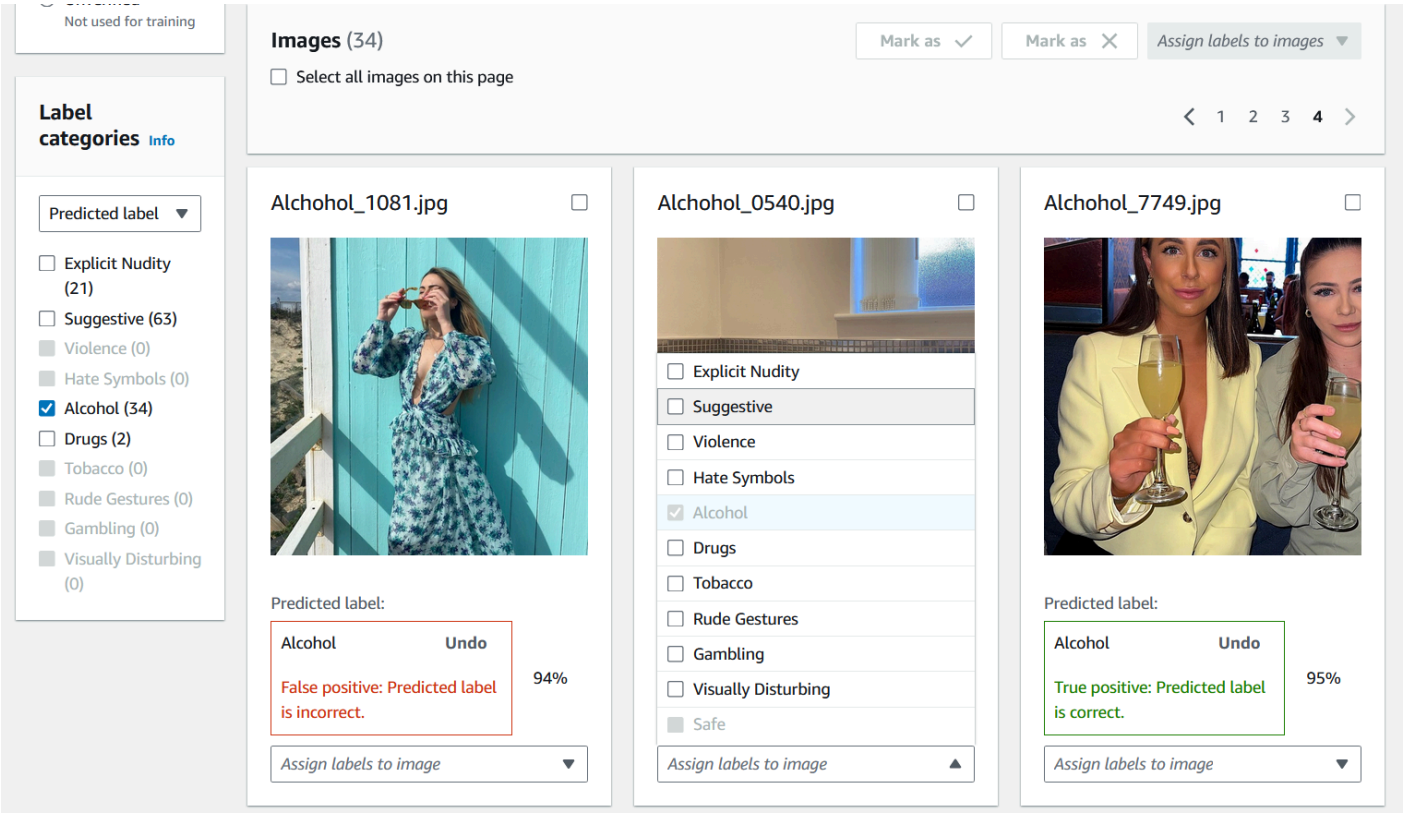
1. 查看基础模型性能后，您需要验证预测。纠正预测结果就可以训练适配器。选择批量分析页面顶部的验证预测。



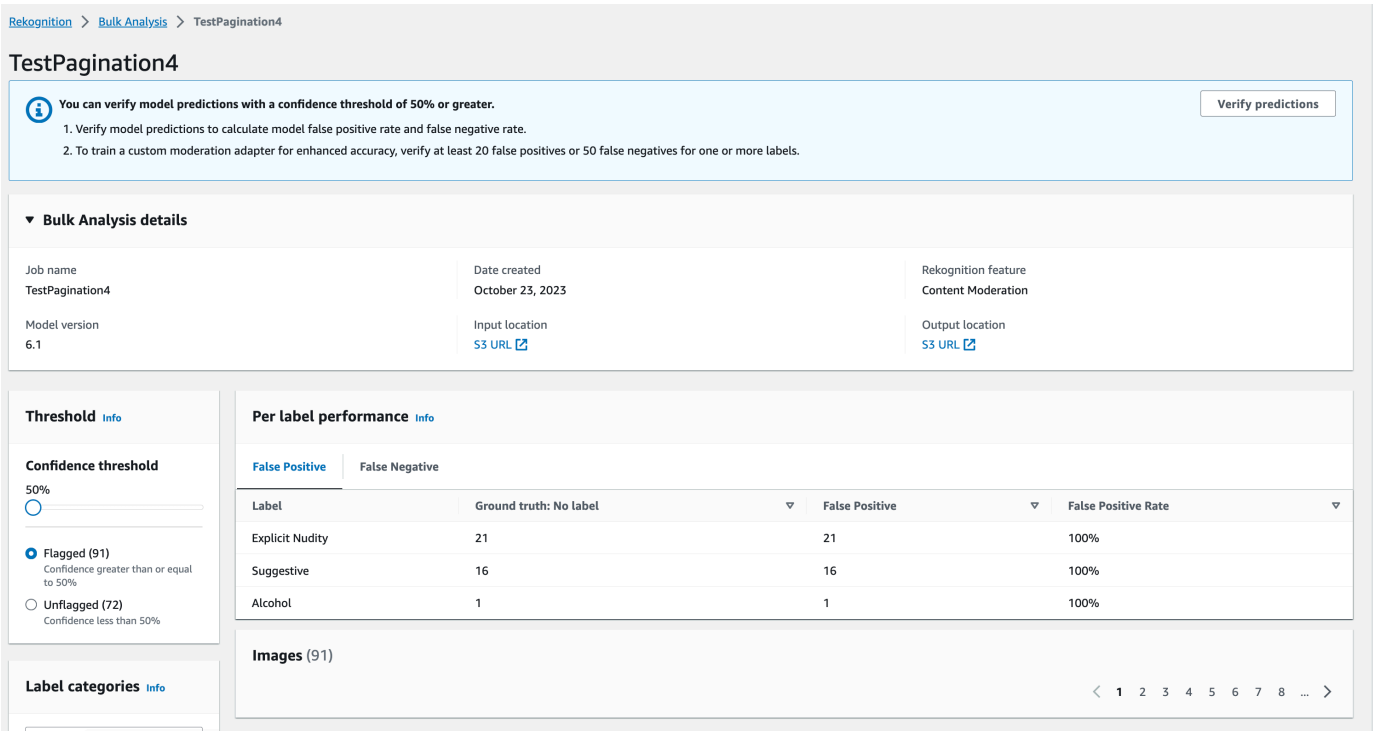
2. 在验证预测页面上，您可以看到您提供给 Rekognition 基础模型或所选适配器的所有图像，以及每张图像的预测标签。您必须使用图像下方的按钮验证每项预测是否正确。使用“X”按钮将预测标记为不正确，使用复选标记按钮将预测标记为正确。要训练适配器，您需要验证给定标签的至少 20 个假阳性预测和 50 个假阴性预测。您验证的预测越多，适配器的性能就越好。



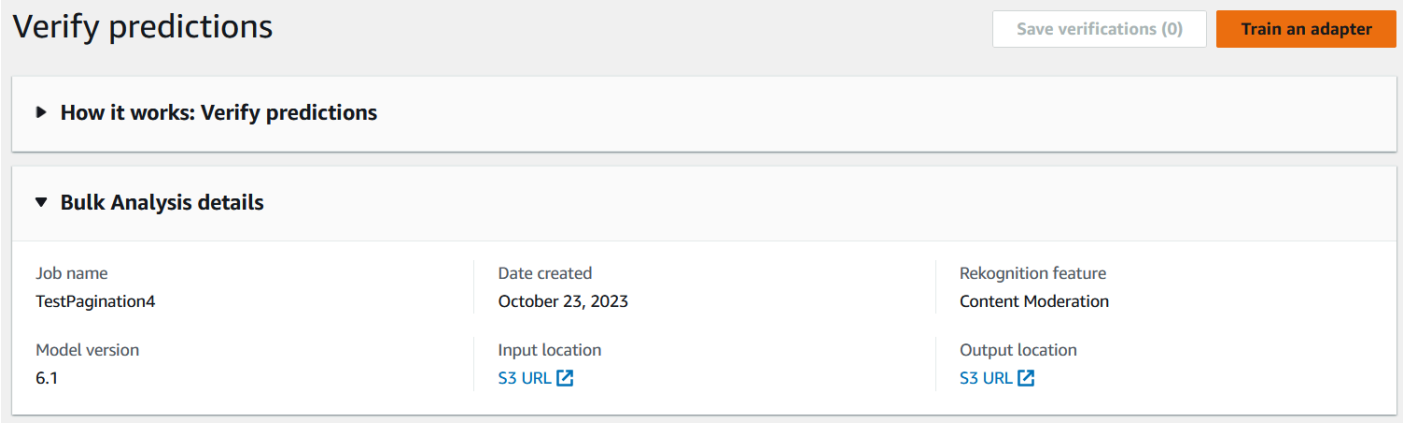
验证预测后，图像下方的文本将发生变化，向您显示已验证的预测类型。验证图像后，您还可以使用为图像分配标签菜单为图像添加其他标签。您可以根据所选的置信度阈值查看模型标记或未标记的图像，或按类别筛选图像。



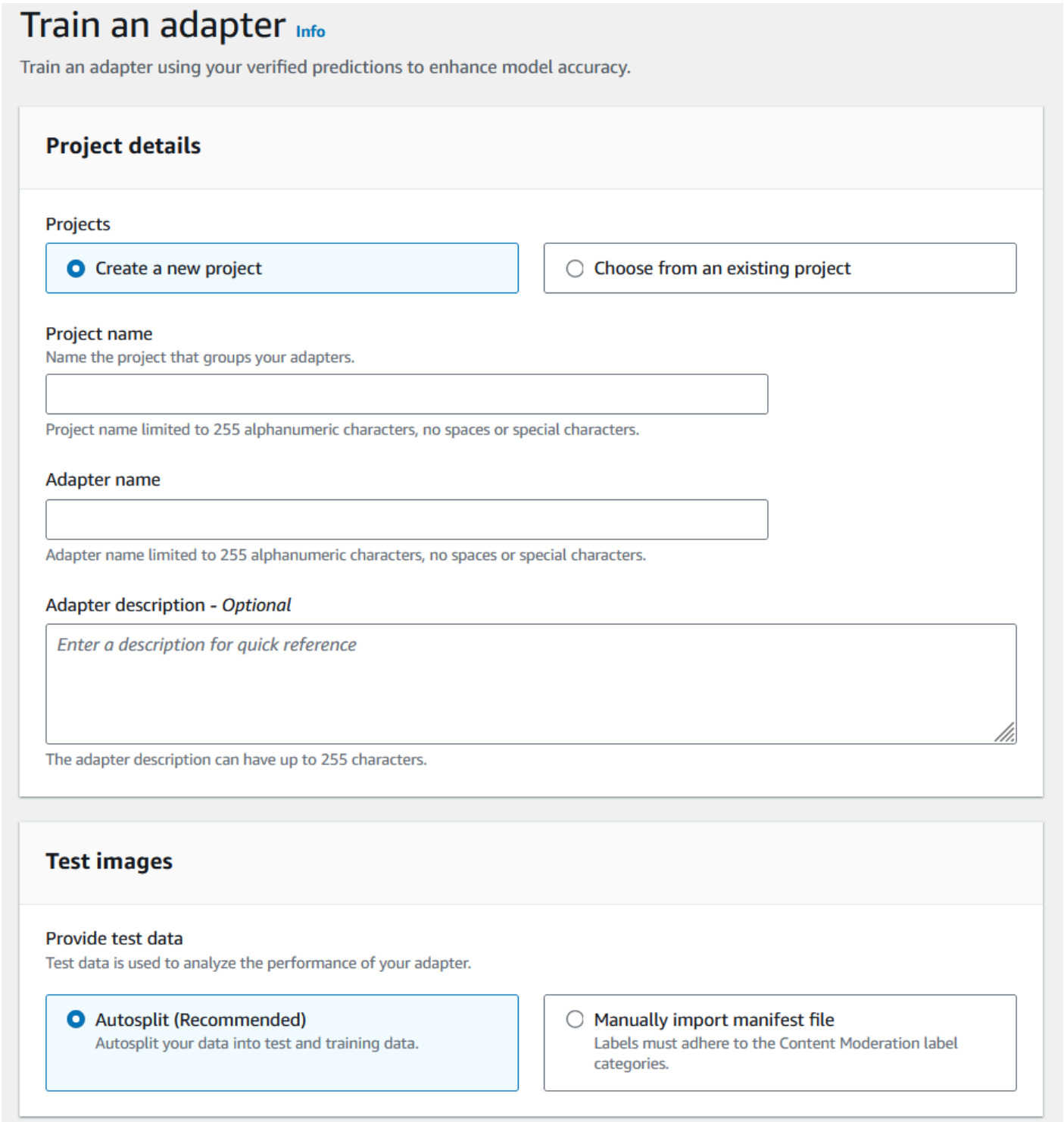
3. 完成验证所有要验证的预测后，您就可以在验证页面的每个标签性能部分看到有关已验证预测的统计数据。您也可以返回批量分析详细信息页面来查看这些统计信息。



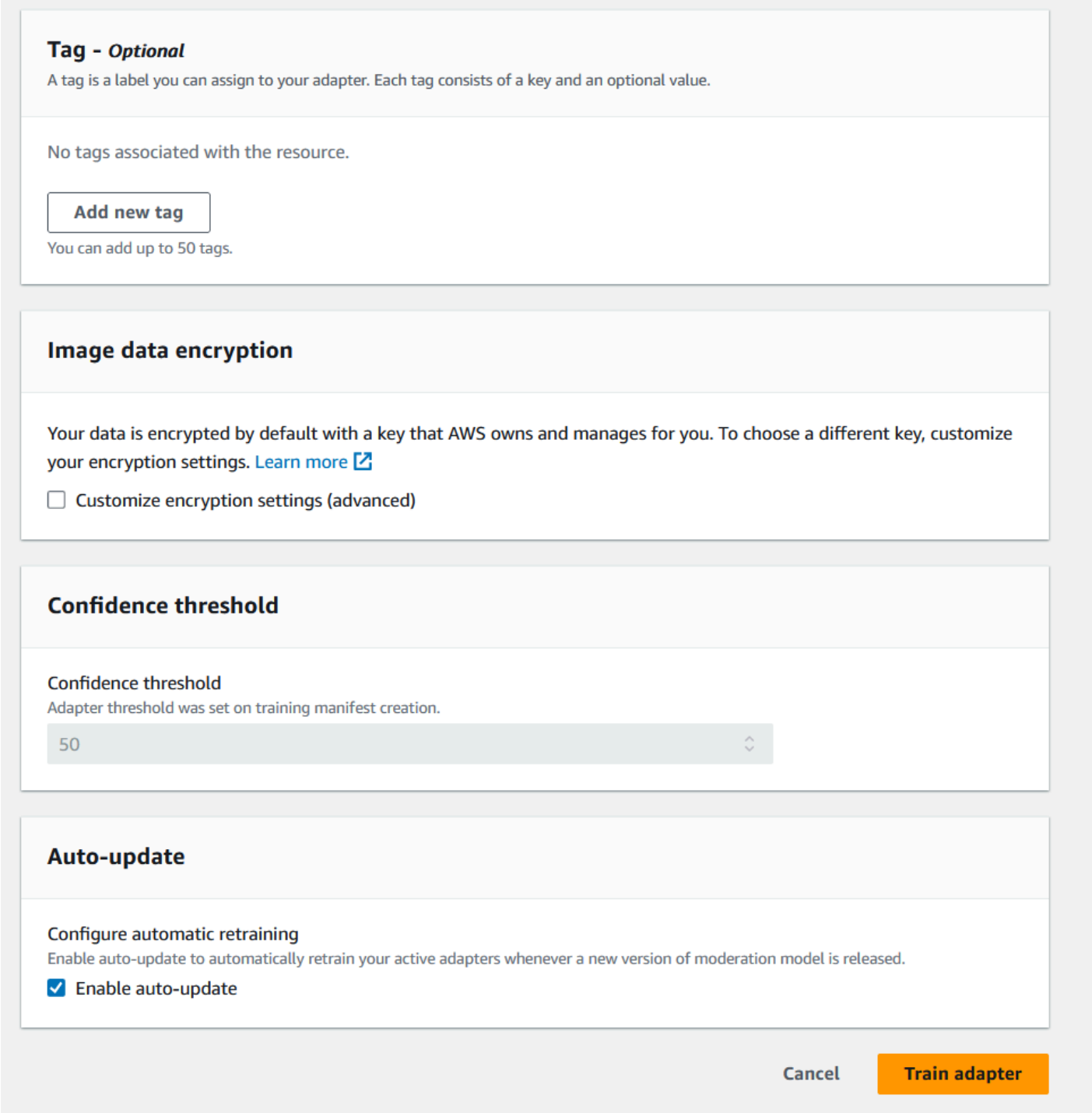
4. 如果您对每个标签性能的统计数据感到满意，请再次转到验证预测页面，然后选择训练适配器按钮开始训练适配器。



5. 在“训练适配器”页面上，系统将提示您创建项目或选择现有项目。命名项目和将包含在项目中的适配器。您还必须指定测试图像的来源。指定图像时，您可以选择自动拆分，让 Rekognition 自动使用部分训练数据作为测试图像，也可以手动指定清单文件。建议选择自动拆分。



6. 指定所需的任何标签，如果您不想使用默认 AWS KMS 密钥，请指定 AWS 密钥。建议保持启用自动更新。
7. 选择训练适配器。

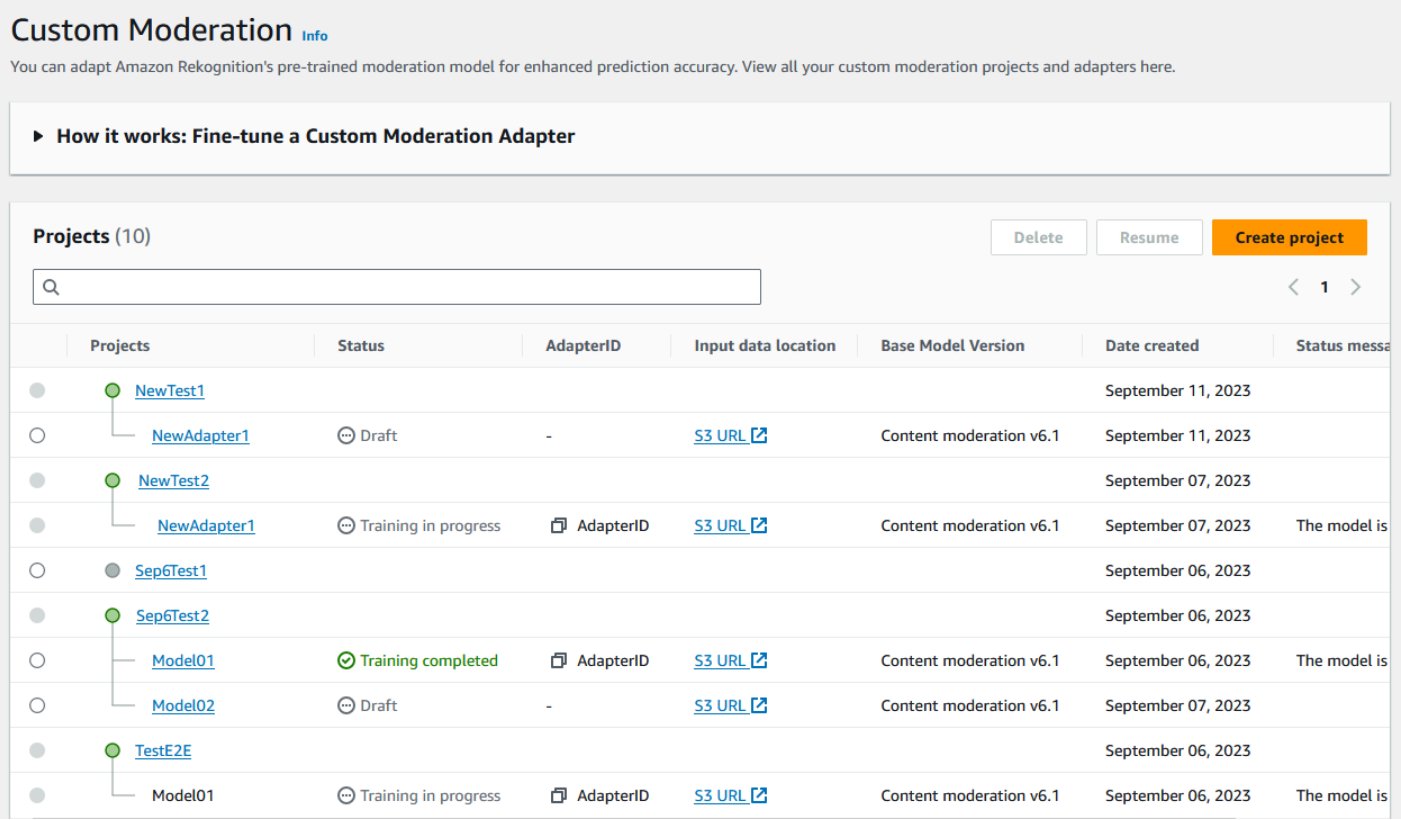


8. 在“自定义审核”登录页面上的适配器状态变为“训练完成”后，您可以查看适配器的性能。请参阅查看适配器性能了解更多信息。

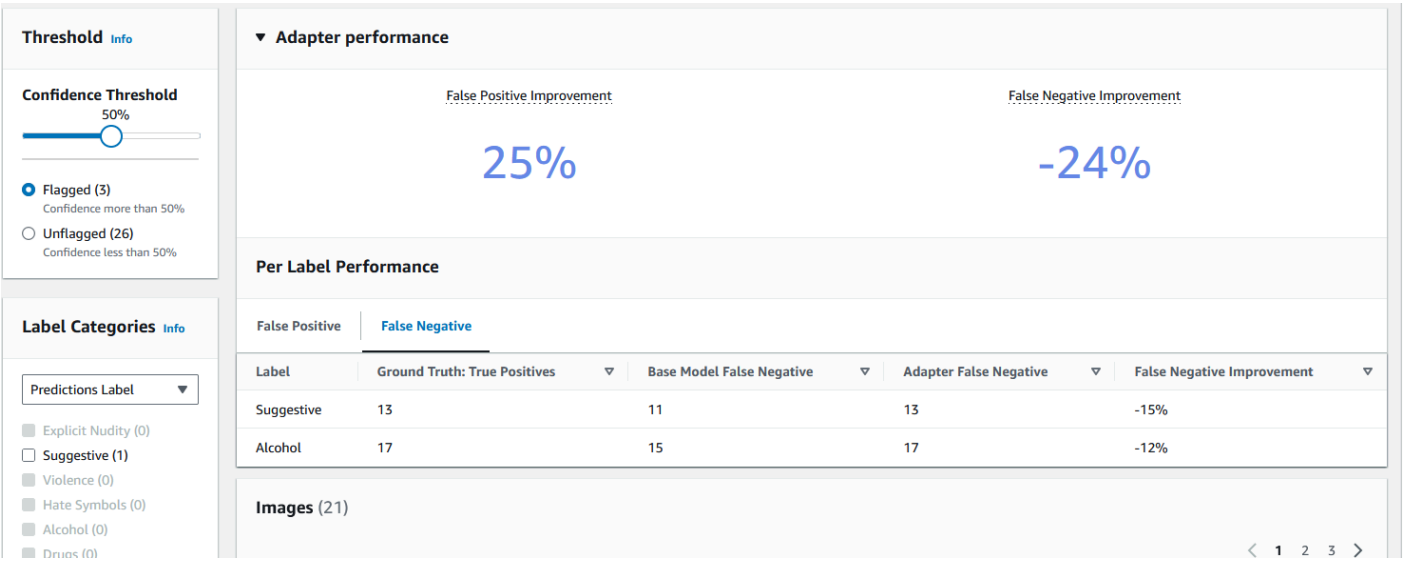
查看适配器性能

要查看适配器性能，请执行以下操作：

1. 使用控制台时，您可以在“自定义审核”登录页面的“项目”选项卡下查看与项目关联的所有适配器的状态。导航到自定义审核登录页面。



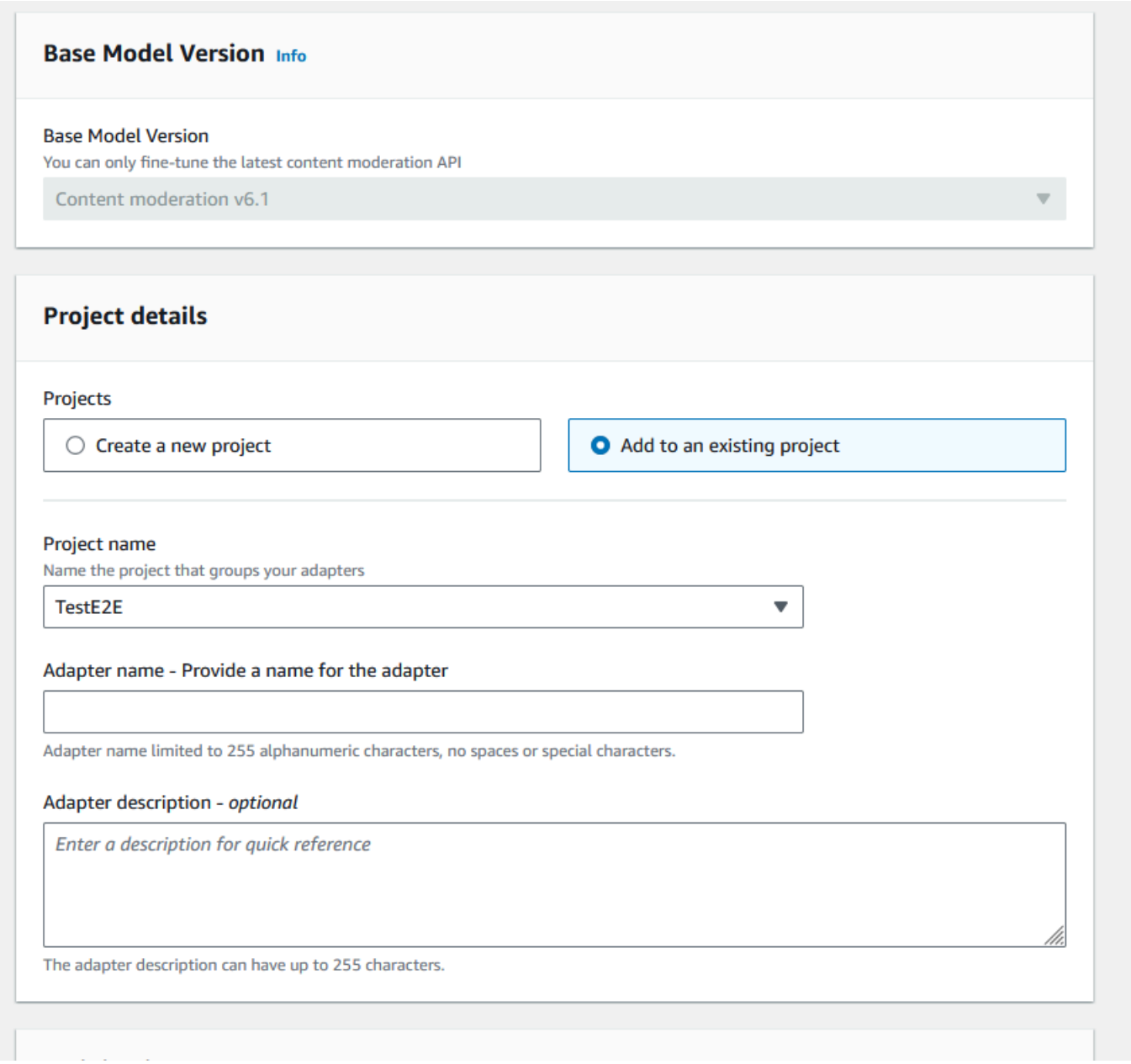
2. 从此列表中选择要查看的适配器。在下面的适配器详细信息页面上，您可以看到适配器的各种指标。



3. 使用阈值面板，您可以更改您的适配器为图像分配标签时必须具备的最小置信度阈值。已标记和未标记实例的数量会随着置信度阈值的调整而变化。您也可以按标签类别进行筛选，以查看所选类别的指标。设置您选择的阈值。
4. 通过检查适配器性能面板中的指标，您可以根据测试数据评测适配器的性能。这些指标的计算方法是将适配器的提取结果与测试集上的“真实情况”注释进行比较。

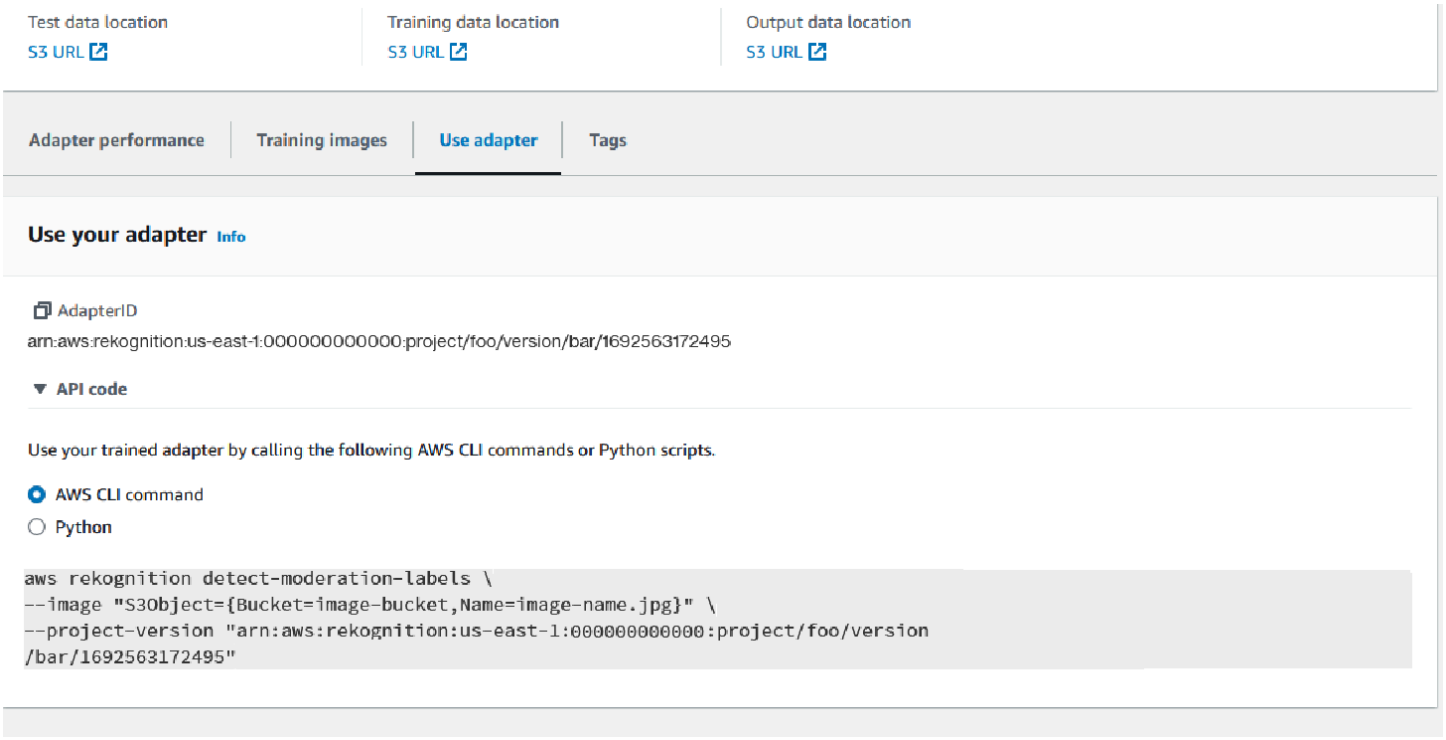
适配器性能面板显示您创建的适配器的假阳性改进率和假阴性改进率。每个标签性能选项卡可用于比较每个标签类别的适配器和基础模型的性能。它显示了基础模型和适配器按标签类别分层的假阳性和假阴性预测计数。通过查看这些指标，您可以确定适配器在哪些方面需要改进。有关这些指标的更多信息，请参阅 评估和改进您的适配器。

为了提高性能，您可以收集更多的训练图像，然后在项目内部创建一个新的适配器。只需返回“自定义审核”登录页面，在项目中创建一个新的适配器，为要训练的适配器提供更多训练图像即可。这次选择添加到现有项目选项，而不是创建新项目，然后从项目名称下拉菜单中选择要在其中创建新适配器的项目。和以前一样，为图片添加注释或提供带有注释的清单文件。



使用适配器

创建适配器后，您可以将其提供给受支持的 Rekognition 操作，例如。DetectModerationLabels要查看可用于对适配器进行推断的代码示例，请选择 “使用适配器” 选项卡，您可以在其中查看 AWS CLI 和 Python 的代码示例。您也可以访问您为其创建适配器的操作文档的相应部分，以查看更多代码示例、设置说明和示例 JSON。



删除您的适配器和项目

您可以删除单个适配器，也可以删除您的项目。您必须先删除与项目关联的所有适配器，然后才能删除项目本身。

1. 要删除与项目关联的适配器，请选择该适配器，然后选择删除。
2. 要删除项目，请选择要删除的项目，然后选择删除。

评估和改进您的适配器

每轮适配器训练结束后，您需要查看 Rekognition 控制台工具中的性能指标，以确定适配器与所需性能水平的接近程度。然后，您可以上传一批新的训练图像并在项目中训练新的适配器，从而进一步提高适配器对图像的准确性。创建适配器的改进版本后，您可以使用控制台删除不再需要的任何较旧版本的适配器。

您也可以使用 DescribeProjectVersionsAPI 操作检索指标。

性能指标

完成训练过程并创建适配器后，请务必评估适配器从图像中提取信息的效果。

Rekognition 控制台中提供了两个指标来帮助您分析适配器的性能：假阳性改进率和假阴性改进率。

您可以通过选择控制台适配器部分的“适配器性能”选项卡来查看任何适配器的这些指标。适配器性能面板显示您创建的适配器的假阳性改进率和假阴性改进率。

假阳性改进衡量适配器识别假阳性的能力比基础模型提高了多少。如果假阳性改进率为 25%，则意味着适配器在测试数据集中对假阳性的识别提高了 25%。

假阴性改进衡量适配器识别假阴性的能力比基础模型提高了多少。如果假阴性改进率为 25%，则意味着适配器在测试数据集中对假阴性的识别提高了 25%。

每个标签性能选项卡可用于比较每个标签类别的适配器和基础模型的性能。它显示了基础模型和适配器按标签类别分层的假阳性和假阴性预测计数。通过查看这些指标，您可以确定适配器在哪些方面需要改进。

例如，如果酒精标签类别的基础模型假阴性率为 15，而适配器假阴性率为 15 或更高，那么在创建新适配器时，就应该重点添加更多包含酒精标签的图像。

使用 Rekognition API 操作时，在调用该操作时会返回 F1-Score 指标。DescribeProjectVersions

改进您的模型

适配器部署是一个迭代过程，因为您可能需要多次训练适配器才能达到目标准确度水平。创建和训练适配器后，您需要测试和评估适配器在各种标签上的性能。

如果您的适配器在任何区域都缺乏准确度，请添加这些图像的新示例，以提高适配器处理这些标签的性能。尝试为适配器提供更多不同的示例，以反映适配器遇到的困难。为您的适配器提供具有代表性的各种图像，使其能够处理各种真实示例。

向训练集添加新图像后，重新训练适配器，然后在测试集和标签上重新评估。重复此过程，直到适配器达到所需的性能水平。如果您提供更具代表性的图像和注释，则在连续的训练迭代中，假阳性分数和假阴性分数将逐渐改善。

清单文件格式

以下各节显示了输入、输出和评估文件的清单文件格式示例。

输入清单

清单文件是一个以 json-line 分隔的文件，每行都包含一个 JSON，其中包含有关单个图像的相关信息。

输入清单中的每个条目都必须包含指向 Amazon S3 存储桶中图像路径的source-ref字段，对于自定义审核，还必须包含带有基本注释的content-moderation-groundtruth字段。一个数据集中的所有图像应位于同一存储桶中。该结构在训练和测试清单文件中都很常见。

针对自定义审核的 CreateProjectVersion 操作使用输入清单中提供的信息来训练适配器。

以下示例是包含单个不安全类的单个图像的清单文件中的一行：

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            }
        ]
   }
}
```

以下示例是包含多个不安全类别（特别是裸露和粗鲁的手势）的单个不安全图像的清单文件中的一行。

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            },
            {
                "Name": "Nudity"
            }
        ]
   }
}
```

以下示例是针对不包含任何不安全类的单个图像的清单文件中的一行：

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": []
```

```
   }
}
```

有关受支持标签的完整列表，请参阅审核内容。

输出清单

训练任务完成后，将返回输出清单文件。输出清单文件是一个以 JSOnLine 分隔的文件，每行都包含一个 JSON，用于保存单个图像的信息。Amazon S3 的路径 OutputManifest 可以从DescribeProjectVersion响应中获得：

- `TrainingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`，用于训练数据集
- `TestingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`，用于测试数据集

输出清单中的每个条目都会返回以下信息：

密钥名称	描述
`source-ref`	引用 s3 中输入清单中提供的图像
`content-moderation-groundtruth`	输入清单中提供的基本事实注释
`detect-moderation-labels`	适配器预测，仅作为测试数据集的一部分
`detect-moderation-labels-base-model`	基础模型预测，仅作为测试数据集的一部分

适配器和基础模型预测以 ConfidenceTrehsold 5.0 格式返回，格式与DetectModerationLabels响应类似。

以下示例显示了适配器和基础模型预测的结构：

```
{
   "ModerationLabels": [
      {
         "Confidence": number,
         "Name": "string",
```

```
          "ParentName": "string"
       }
    ],
    "ModerationModelVersion": "string",
    "ProjectVersion": "string"
}
```

有关返回的标签的完整列表，请参阅审核内容。

评估结果清单

训练任务完成后，将返回评估结果清单文件。评估结果清单是训练任务输出的 JSON 文件，其中包含有关适配器在测试数据上的表现的信息。

Amazon S3 评估结果清单的路径可以从 DescribeProejctVersion 响应中的`EvaluationResult.Summary.S3Object`字段获取。

以下示例显示评估结果清单的结构：

```
{
    "AggregatedEvaluationResults": {
       "F1Score": number
    },

    "EvaluationDetails": {
        "EvaluationEndTimestamp": "datetime",
        "Labels": [
            "string"
        ],
        "NumberOfTestingImages": number,
        "NumberOfTrainingImages": number,
        "ProjectVersionArn": "string"
    },

    "ContentModeration": {
        "InputConfidenceThresholdEvalResults": {
            "ConfidenceThreshold": float,
            "AggregatedEvaluationResults": {
                "BaseModel": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
```

```
                },
                "Adapter": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                }
            },
            "LabelEvaluationResults": [
                {
                    "Label": "string",
                    "BaseModel": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    },
                    "Adapter": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    }
                }
            ]
        }
        "AllConfidenceThresholdsEvalResults": [
            {
                "ConfidenceThreshold": float,
                "AggregatedEvaluationResults": {
                    "BaseModel": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    },
                    "Adapter": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    }
                },
                "LabelEvaluationResults": [
```

```
                    {
                       "Label": "string",
                        "BaseModel": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        },
                        "Adapter": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        }
                    }
                ]
            }
        ]
    }
}
```

评估清单文件包含：

- 汇总结果由 F1Score 定义
- 评估任务的详细信息包括训练图像的数量 ProjectVersionArn、测试图像的数量以及适配器接受过训练的标签。
- 基础模型和适配器性能的汇总 TruePositive TrueNegative FalsePositive、、和 FalseNegative 结果。
- 每个标签 TruePositive、 TrueNegative FalsePositive、以及基本模型和适配器性能的 FalseNegative 结果，在输入置信度阈值下计算。
- 在不同的置信度阈值 TrueNegative下 TruePositive FalsePositive，基础模型和适配器性能的汇总 FalseNegative 结果和每个标签、、以及结果。置信度阈值从 5 到 100 不等，以 5 为单位。

训练适配器的最佳实践

建议您在创建、训练和使用适配器时遵守以下最佳实践：

1. 样本图像数据应捕捉客户打算禁止的代表性错误。如果模型在视觉相似的图像上反复犯错误，请务必带上其中许多图像进行训练。
2. 与其只引入模型在特定审核标签上犯错的图片，还要确保引入模型在该审核标签上没有犯错的图片。
3. 提供至少 50 个假阴性样本或 20 个假阳性样本用于训练，以及至少 20 个样本进行测试。但是，为了提高适配器性能，请尽可能多地提供带注释的图像。
4. 为所有图像注释对您重要的所有标签 — 如果您决定需要为图像上的某个标签的引用添加注释，请确保为所有其他图像上的该标签的引用进行注释。
5. 样本图像数据应在标签上包含尽可能多的变体，重点是代表将在生产环境中分析的图像的实例。

设置 AutoUpdate权限

Rekognition 支持 AutoUpdate 自定义适配器的功能。这意味着，当项目启用 AutoUpdate 标记时，会尽最大努力尝试自动再训练。这些自动更新需要访问您的培训/测试数据集的权限以及用于训练客户 AWS KMS 适配器的密钥。您可以按照以下步骤提供这些权限。

Amazon S3 存储桶权限

默认情况下，所有 Amazon S3 存储桶和对象都是私有的。只有资源所有者，即创建存储桶的 AWS 账户，才能访问存储桶及其包含的任何对象。不过，资源所有者可以选择通过编写存储桶策略来向其他资源和用户授予访问权限。

如果您想在自定义适配器训练中创建或修改 AmazonS3 存储桶以用作输入数据集的来源和训练结果的目标，则必须进一步修改存储桶策略。要读取或写入 Amazon S3 存储桶，Rekognition 必须具有以下权限。

Rekognition 需要 Amazon S3 策略

Rekognition 需要具有以下属性的权限策略：

- 语句 SID
- 存储桶名称
- Rekognition 的服务主体名称。
- Rekognition 存储桶及其所有内容所需的资源
- Rekognition 需要采取的必要行动。

以下策略允许 Rekognition 在自动再训练期间访问 Amazon S3 存储桶。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "AllowRekognitionAutoUpdateActions",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:HeadObject",
                "s3:HeadBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

您可以按照本指南将上述存储桶策略添加到您的 S3 存储桶。

在此处查看有关存储桶策略的更多信息。

AWS KMS 密钥权限

Rekognition 允许您在训练自定义适配器时提供可 KmsKeyId 选适配器。如果提供，Rekognition 将使用此密钥对复制到服务中的训练和测试图像进行加密，用于训练模型。该密钥还用于加密写入输出 Amazon S3 存储桶 (OutputConfig) 的训练结果和清单文件。

如果您选择提供 KMS 密钥作为自定义适配器训练的输入（即 `Rekognition:CreateProjectVersion`），则必须进一步修改 KMS 密钥政策，以允许 Rekognition 服务主体将来使用此密钥进行自动再训练。Rekognition 必须具有以下权限。

Rekognition 必填密钥策略 AWS KMS

Amazon Rekognition 需要具有以下属性的权限策略：

- 语句 SID
- Amazon Rekognition 的服务主体名称。
- Amazon Rekognition 需要采取的必要行动。

以下密钥政策允许 Amazon Rekognition 在自动再训练期间访问 Amazon KMS 密钥：

```
{
  "Version": "2023-10-06",
  "Statement": [
    {
      "Sid": "KeyPermissions",
      "Effect": "Allow",
      "Principal": {
            "Service": "rekognition.amazonaws.com"
        },
      "Action": [
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ]
      "Resource": "*"
    }
  ]
}
```

您可以按照本指南将上述 AWS KMS 政策添加到您的 AWS KMS 密钥中。

在此处查看有关 AWS KMS 政策的更多信息。

AWS Rekognition 的健康控制面板通知

您的 AWS 健康控制面板为来自 Rekognition 的通知提供支持。就可能影响您的应用程序的 Rekognition 模型的计划更改，这些通知可以让您注意相关情况并提供修正指导。当前只有特定于 Rekognition 内容审核功能的事件可用。

AWS 健康控制面板是 AWS 健康服务的一部分。它不需要设置，您的账户中通过身份验证的任何用户都可以查看。有关更多信息，请参阅 AWS Health Dashboard 入门。

如果您收到类似于下文的通知消息，则应将其视为警报并采取措施。

通知示例：Rekognition 内容审核提供了新的模型版本。

Rekognition 将该AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION事件发布到 Healt AWS h Dashboard，表示审核模型的新版本已经发布。如果您在此 API 中使用 DetectModerationLabels API 和适配器，则此事件非常重要。新模型可能会影响质量，具体取决于您的用例，并且最终会取代以前的模型版本。建议您在收到此警报时验证您的模型质量并注意模型更新时间表。

如果您收到模型版本更新通知，则应将其视为警报以采取行动。如果您不使用适配器，则应根据现有用例评估更新后的模型的质量。如果您使用适配器，则应使用更新的模型训练新的适配器并评估其质量。如果您有自动训练集，则会自动训练新的适配器，然后您可以评估其质量。

```
{
   "version": "0",
    "id": "id-number",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-10-06T06:27:57Z",
    "region": "region",
    "resources": [],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/
AWS_MODERATION_MODEL_UPDATE_NOTIFICATION_event-number",
        "service": "Rekognition",
        "eventTypeCode": "AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION",
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId": "communication-id-number",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "lastUpdatedTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "statusCode": "open",
        "eventRegion": "us-east-1",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "A new model version is available for Rekognition
 Content Moderation."
            }
        ]
    }
}
```

请参阅使用 A mazon 监控 AWS Health 事件，使用 EventBridge来检测和应对 AWS 健康事件 EventBridge。

使用 Amazon Augmented AI 审核不当内容

通过 Amazon Augmented AI (Amazon A2I)，您可以构建人工审核机器学习预测所需的工作流程。

Amazon Rekognition 可直接与 Amazon A2I 集成，以便您轻松实施人工审核，检测出不安全图像的使用案例。Amazon A2I 提供了用于图像监管的人工审核工作流程。这样一来，您可以从 Amazon Rekognition 轻松审核预测。您可以为使用案例定义置信度阈值，并随着时间的推移进行调整。借助 Amazon A2I，您可以使用自己的组织或 Amazon Mechanical Turk 中的审核人员池。您还可以使用经 AWS 预先筛选的供应商队伍，以确保质量和遵守安全程序。

以下步骤将指导您如何设置 Amazon A2I 和 Amazon Rekognition。首先，使用 Amazon A2I 创建具有人工审核触发条件的流定义。然后，将流定义的 Amazon 资源名称 (ARN) 传递给 Amazon Rekognition `DetectModerationLabel` 操作。在 `DetectModerationLabel` 响应中，可以看到是否需要人工审核。人工审核的结果位于流定义设置的 Amazon S3 存储桶中。

要观看如何将亚马逊 A2I 与 Amazon Rekognition 配合使用的 end-to-end演示，请参阅亚马逊 AI 开发者指南中的以下教程之一。 SageMaker

- 演示：开始使用 Amazon A2I 控制台
- 演示：开始使用 Amazon A2I API

  要开始使用 API，您还可以运行示例 Jupyter 笔记本。参见将 SageMaker 笔记本实例与亚马逊 A2I Jupyter Notebook 配合使用，在人工智能笔记本实例中使用与亚马逊 Rekognition [示例] 集成的笔记本亚马逊增强人工智能（亚马逊 A2I）[示例]。 SageMaker

DetectModerationLabels 使用亚马逊 A2I 运行

> **Note**
>
> 在同一 AWS 区域中创建所有 Amazon A2I 资源和 Amazon Rekognition 资源。

1. 完成 AI 文档中 Amazon Agumented AI 入门中列出的先决条件。SageMaker

   此外，请记得按照 AI 文档中的 A mazon Agumented AI 中的权限和安全页面设置您的 I SageMaker A M 权限。

2. 按照 SageMaker AI 文档中有关创建人工审阅工作流程的说明进行操作。

   人工审核工作流程用于管理图像的处理。它包含触发人工审核的条件、图像发送到的工作团队、工作团队使用的 UI 模板，以及工作团队的结果发送到的 Amazon S3 存储桶。

   在CreateFlowDefinition通话中，您需要将设置HumanLoopRequestSource为 “AWS/Rekognition/DetectModerationLabels/Image/V3”。之后，您需要决定希望如何设置人工审核的触发条件。

   Amazon Rekognition 为 ConditionType 提供了两个选项：ModerationLabelConfidenceCheck 和 Sampling。

   当审核标签的置信度在一定范围内时，ModerationLabelConfidenceCheck 会创建人工循环。最后，Sampling 会发送随机比例的已处理文档供人工审核。每个 ConditionType 会使用一组不同的 ConditionParameters 来设置人工审核中的结果。

   ModerationLabelConfidenceCheck 具有 ConditionParameters ModerationLableName，用于设置需要人工审核的键。此外，它还具有置信度，它设定了发送给人工审核的百分比范围，设置了 LessThan GreaterThan、和 Equals。SamplingRandomSamplingPercentage它设定了将发送给人工审查的文档的百分比。

   以下代码示例是 CreateFlowDefinition 的部分调用。如果图像在标签“暗示”上的评级低于 98%，并且在标签“女性泳装或内衣”上的评级高于 95%，则该图像将发送到人工审核。这意味着，如果图像不被认为是暗示性的，但确实有穿着内衣或泳装的女性，您可以使用人工审核再次确认检查图像。

```
def create_flow_definition():
'''
Creates a Flow Definition resource

Returns:
struct: FlowDefinitionArn
'''
humanLoopActivationConditions = json.dumps(
    {
        "Conditions": [
            {
              "And": [
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
```

```
                        "ConditionParameters": {
                            "ModerationLabelName": "Suggestive",
                            "ConfidenceLessThan": 98
                        }
                    },
                    {
                        "ConditionType": "ModerationLabelConfidenceCheck",
                        "ConditionParameters": {
                            "ModerationLabelName": "Female Swimwear Or Underwear",
                            "ConfidenceGreaterThan": 95
                        }
                    }
                ]
            }
        ]
    }
)
```

CreateFlowDefinition 将返回 FlowDefinitionArn，您在下一步中调用 DetectModerationLabels 时可使用该 ARN。

有关更多信息，请参阅 SageMaker AI API 参考CreateFlowDefinition中的。

3. 调用 DetectModerationLabels 时，请设置 HumanLoopConfig 参数，如检测不当图像中所述。有关带有 HumanLoopConfig 设置的 DetectModerationLabels 调用示例，请参阅步骤 4。

   a. 在 HumanLoopConfig 参数中，将 FlowDefinitionArn 设置为在步骤 2 中创建的流定义的 ARN。

   b. 设置 HumanLoopName。此名称在区域内应该是唯一的，并且必须采用小写。

   c. （可选）可以使用 DataAttributes 参数，设置传递到 Amazon Rekognition 的图像是否不包含个人身份信息。必须设置此参数才能将信息发送到 Amazon Mechanical Turk。

4. 运行 DetectModerationLabels。

   以下示例说明如何使用 AWS CLI 和在 s 适用于 Python (Boto3) 的 AWS SDK e DetectModerationLabels HumanLoopConfig t 中运行。

适用于 Python (Boto3) 的 AWS SDK

以下请求示例使用 SDK for Python (Boto3)。有关更多信息，请参阅《AWS SDK for Python (Boto) API 参考》中的 detect_moderation_labels。

```
import boto3

rekognition = boto3.client("rekognition", aws-region)

response = rekognition.detect_moderation_labels( \
        Image={'S3Object': {'Bucket': bucket_name, 'Name': image_name}}, \
        HumanLoopConfig={ \
            'HumanLoopName': 'human_loop_name', \
            'FlowDefinitionArn': , "arn:aws:sagemaker:aws-
region:aws_account_number:flow-definition/flow_def_name" \
            'DataAttributes': {'ContentClassifiers':
 ['FreeOfPersonallyIdentifiableInformation','FreeOfAdultContent']}
         })
```

AWS CLI

以下请求示例使用 AWS CLI。有关更多信息，请参阅 AWS CLI 命令参考 中的 detect-moderation-labels。

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config
 HumanLoopName="human_loop_name",FlowDefinitionArn="arn:aws:sagemaker:aws-
region:aws_account_number:flow-
definition/
flow_def_name",DataAttributes='{ContentClassifiers=["FreeOfPersonallyIdentifiableInforma
 "FreeOfAdultContent"]}'
```

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config \
        '{"HumanLoopName": "human_loop_name", "FlowDefinitionArn":
 "arn:aws:sagemaker:aws-region:aws_account_number:flow-
definition/flow_def_name", "DataAttributes": {"ContentClassifiers":
 ["FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"]}}'
```

当你在HumanLoopConfig启用状态下运行DetectModerationLabels时，Amazon Rekognition 会调用 AI API 操作 SageMaker 。StartHumanLoop此命令从 DetectModerationLabels 获取响应，并根据示例中的流定义条件对其进行检查。如果符合审核条件，则返回 HumanLoopArn。这表示您在流程定义中设置的工作团队成员现在可以审核该图像。调用 Amazon Augmented AI 运行时系统操作 DescribeHumanLoop 可提供有关循环的结果的信息。有关更多信息，请参阅 Amazon A DescribeHumanLoopgumented AI API 参考文档。

图像经审核后，可以在流定义输出路径中指定的存储桶中看到结果。审核完成后，亚马逊 A2I 还会通过亚马逊 CloudWatch 活动通知您。要查看要查找的事件，请参阅 SageMaker AI 文档中的CloudWatch 事件。

有关更多信息，请参阅 A I 文档中的 Amazon Agumented SageMaker AI 入门。

检测文本

Amazon Rekognition 可以检测图像和视频中的文本。然后，它会将检测到的文本转换为机器可读的文本。您可以使用图像中的机器可读文本检测来实施一些解决方案，例如：

- 可视化搜索。例如，检索和显示包含相同文本的图像。
- 内容见解。例如，提供对文本中出现的主题的见解，该文本是在提取的视频帧中识别的。您的应用程序可以搜索识别的文本中的相关内容，例如新闻、体育赛事比分、运动员号码和字幕。
- 导航。例如，为视障人士开发可识别餐厅、商店或路标中的名称的支持语音的移动应用程序。
- 公共安全和交通运输支持。例如，从路况摄像头图像中检测车牌号码。
- 过滤。例如，从图像中过滤掉个人身份信息 (PII)。

对于视频中的文本检测，您可以实施一些解决方案，例如：

- 在视频中搜索包含特定文本关键字的剪辑，例如，新闻节目中图片上的来宾姓名。
- 通过检测意外文本、亵渎性词汇或垃圾邮件，审核内容是否符合组织标准。
- 查找视频时间线上的所有文本叠加以进行进一步处理，例如替换文本为另一种语言的文本以实现内容国际化。
- 查找文本位置，以便其他图形能够相应地对齐。

要检测 JPEG 或 PNG 格式图像中的文本，请使用DetectText操作。要异步检测视频中的文本，请使用StartTextDetection和GetTextDetection操作。图像和视频文本检测操作支持大多数字体，包括高度风格化的字体。在检测文本后，Amazon Rekognition 将创建检测到的单词和文本行的表示形式，显示它们之间的关系，并告知您文本在图像或视频帧中的位置。

`DetectText` 和 `GetTextDetection` 操作可以检测单词和行。单词 是一个或多个不用空格分隔的字母字符。`DetectText` 可以在一张图片中检测多达 100 个单词。`GetTextDetection` 每帧视频还可以检测多达 100 个单词。

单词是一个或多个字母字符，不用空格分隔。Amazon Rekognition 旨在检测英语、阿拉伯语、俄语、德语、法语、意大利语、葡萄牙语和西班牙语的单词。

行 是一个由等间距单词组成的字符串。一行不一定是一个完整的句子（句号不表示一行的结尾）。例如，Amazon Rekognition 将一个驾照号码检测为一行。当后面没有对齐的文本或单词之间有较大的间距（相对于单词的长度）时，一行便结束了。根据单词之间的间距，Amazon Rekognition 可能在向同一方向对齐的文本中检测到多个行。如果一个句子跨多个行，则操作将返回多个行。

请考虑以下图像。

蓝色框表示有关 DetectText 操作返回的检测到的文本和该文本的位置的信息。在此示例中，Amazon Rekognition 将“IT's”、“MONDAY”、“but”、“keep”和“Smiling”作为单词检测。Amazon Rekognition 将“IT's”、“MONDAY”、“but keep”和“Smiling”检测为一行。文本必须在横轴的 +/- 90 度方向以内才能被检测到。

有关示例，请参阅检测图像中的文本。

主题

- 检测图像中的文本
- 检测存储视频中的文本

检测图像中的文本

您可以提供输入图像作为图像字节数组（base64 编码的图像字节）或 Amazon S3 对象。在此过程中，您将 JPEG 或 PNG 图像上传到您的 S3 存储桶并指定文件名称。

检测图像中的文本 (API)

1. 如果您尚未完成，请完成以下先决条件。

   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。

   b. 安装并配置 AWS Command Line Interface 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将包含文本的图像上传到您的 S3 存储桶。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下示例调用 DetectText 操作。

   Java

   以下示例代码显示在图像中检测到的行和单词。

   将amzn-s3-demo-bucket和photo的值替换为您在步骤 2 中使用的 S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.DetectTextRequest;
import com.amazonaws.services.rekognition.model.DetectTextResult;
import com.amazonaws.services.rekognition.model.TextDetection;
import java.util.List;



public class DetectText {

   public static void main(String[] args) throws Exception {
```

```
      String photo = "inputtext.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();



      DetectTextRequest request = new DetectTextRequest()
              .withImage(new Image()
              .withS3Object(new S3Object()
              .withName(photo)
              .withBucket(bucket)));


      try {
         DetectTextResult result = rekognitionClient.detectText(request);
         List<TextDetection> textDetections = result.getTextDetections();

         System.out.println("Detected lines and words for " + photo);
         for (TextDetection text: textDetections) {

                 System.out.println("Detected: " + text.getDetectedText());
                 System.out.println("Confidence: " +
 text.getConfidence().toString());
                 System.out.println("Id : " + text.getId());
                 System.out.println("Parent Id: " + text.getParentId());
                 System.out.println("Type: " + text.getType());
                 System.out.println();
         }
      } catch(AmazonRekognitionException e) {
         e.printStackTrace();
      }
   }
}
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
/**
```

```
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

//snippet-start:[rekognition.java2.detect_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DetectTextRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectTextResponse;
import software.amazon.awssdk.services.rekognition.model.TextDetection;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextImage {

 public static void main(String[] args) {

     final String usage = "\n" +
```

```
        "Usage: " +
        "   <sourceImage>\n\n" +
        "Where:\n" +
        "   sourceImage - The path to the image that contains text (for
example, C:\\AWS\\pic1.png). \n\n";

  if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String sourceImage = args[0] ;
    Region region = Region.US_WEST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("default"))
        .build();

    detectTextLabels(rekClient, sourceImage );
    rekClient.close();
}

// snippet-start:[rekognition.java2.detect_text.main]
public static void detectTextLabels(RekognitionClient rekClient, String
sourceImage) {

    try {
        InputStream sourceStream = new FileInputStream(sourceImage);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
        Image souImage = Image.builder()
            .bytes(sourceBytes)
            .build();

        DetectTextRequest textRequest = DetectTextRequest.builder()
            .image(souImage)
            .build();

        DetectTextResponse textResponse = rekClient.detectText(textRequest);
        List<TextDetection> textCollection = textResponse.textDetections();
        System.out.println("Detected lines and words");
        for (TextDetection text: textCollection) {
            System.out.println("Detected: " + text.detectedText());
            System.out.println("Confidence: " + text.confidence().toString());
            System.out.println("Id : " + text.id());
```

```
            System.out.println("Parent Id: " + text.parentId());
            System.out.println("Type: " + text.type());
            System.out.println();
        }

    } catch (RekognitionException | FileNotFoundException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
// snippet-end:[rekognition.java2.detect_text.main]
```

AWS CLI

此 AWS CLI 命令显示 detect-text CLI 操作的 JSON 输出。

将amzn-s3-demo-bucket和Name的值替换为您在步骤 2 中使用的 S3 存储桶和图像的名称。

将profile_name的值替换为您的开发人员资料的名称。

```
aws rekognition detect-text  --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket","Name":"image-name"}}" --profile default
```

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。例如，请参阅以下内容：

```
aws rekognition detect-text  --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile default
```

Python

以下示例代码显示在图像中检测到的行和单词。

将amzn-s3-demo-bucket和photo的值替换为您在步骤 2 中使用的 S3 存储桶和图像的名称。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
# Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import boto3

def detect_text(photo, bucket):

    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')

    response = client.detect_text(Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}})

    textDetections = response['TextDetections']
    print('Detected text\n----------')
    for text in textDetections:
        print('Detected text:' + text['DetectedText'])
        print('Confidence: ' + "{:.2f}".format(text['Confidence']) + "%")
        print('Id: {}'.format(text['Id']))
        if 'ParentId' in text:
            print('Parent Id: {}'.format(text['ParentId']))
        print('Type:' + text['Type'])
        print()
    return len(textDetections)

def main():
    bucket = 'amzn-s3-demo-bucket'
    photo = 'photo-name'
    text_count = detect_text(photo, bucket)
    print("Text detected: " + str(text_count))

if __name__ == "__main__":
    main()
```

.NET

以下示例代码显示在图像中检测到的行和单词。

将amzn-s3-demo-bucket和photo的值替换为您在步骤 2 中使用的 S3 存储桶和图像的名称。

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
```

```
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectText
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "amzn-s3-demo-bucket";

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient();

        DetectTextRequest detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                }
            }
        };

        try
        {
            DetectTextResponse detectTextResponse =
 rekognitionClient.DetectText(detectTextRequest);
            Console.WriteLine("Detected lines and words for " + photo);
            foreach (TextDetection text in detectTextResponse.TextDetections)
            {
                Console.WriteLine("Detected: " + text.DetectedText);
                Console.WriteLine("Confidence: " + text.Confidence);
                Console.WriteLine("Id : " + text.Id);
                Console.WriteLine("Parent Id: " + text.ParentId);
                Console.WriteLine("Type: " + text.Type);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
```

```
}
```

Node.JS

以下示例代码显示在图像中检测到的行和单词。

将amzn-s3-demo-bucket和photo的值替换为您在步骤 2 中使用的 S3 存储桶和图像的名称。将region的值替换为在您的 .aws 凭证中找到的区域。将创建 Rekognition 会话的行中的profile_name值替换为您的开发人员资料的名称。

```
var AWS = require('aws-sdk');

const bucket = 'bucket' // the bucketname without s3://
const photo  = 'photo' // the name of file

const config = new AWS.Config({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
})
AWS.config.update({region:'region'});
const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
}
client.detectText(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // handle error if an error occurred
  } else {
    console.log(`Detected Text for: ${photo}`)
    console.log(response)
    response.TextDetections.forEach(label => {
      console.log(`Detected Text: ${label.DetectedText}`),
      console.log(`Type: ${label.Type}`),
      console.log(`ID: ${label.Id}`),
      console.log(`Parent ID: ${label.ParentId}`),
      console.log(`Confidence: ${label.Confidence}`),
      console.log(`Polygon: `)
      console.log(label.Geometry.Polygon)
```

```
    }
    )
  }
});
```

DetectText 操作请求

在 DetectText 操作中，您可将输入图像作为 base64 编码的字节数组提供，也可将其作为 Amazon S3 存储桶中存储的图像提供。以下示例 JSON 请求显示从 Amazon S3 存储桶加载的图像。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "inputtext.jpg"
        }
    }
}
```

筛选条件

通过按文本区域、大小和置信度评分进行筛选，您可以更加灵活地控制文本检测输出。通过使用感兴趣的区域，您可以轻松地将文本检测限制到与您相关的区域，例如，在从机器的图像中读取零件编号时，档案照片的右上角或相对于参考点的固定位置。单词边界框大小筛选器可用于避免噪点或不相关的小背景文本。通过单词置信度筛选器，您可以移除因模糊或脏污导致的不可靠的结果。

有关筛选值的信息，请参阅 DetectTextFilters。

您可以使用以下筛选器：

- MinConfidence— 设置字词检测的置信度。从结果中排除检测置信度低于此级别的单词。值应介于 0 和 100 之间。
- MinBoundingBoxWidth— 设置单词边界框的最小宽度。将从结果中排除其边界框小于此值的单词。该值是相对于图像帧宽度的。
- MinBoundingBoxHeight— 设置单词边框的最小高度。将从结果中排除其边界框高度小于此值的单词。该值是相对于图像帧高度的。
- RegionsOfInterest— 将检测范围限制在图像帧的特定区域。这些值是相对于帧的尺寸的。对于仅部分位于区域内的文本，响应是不确定的。

DetectText 操作响应

该DetectText操作分析图像并返回一个数组 TextDetections，其中每个元素 (TextDetection) 表示图像中检测到的行或字。对于每个元素，DetectText 将返回以下信息：

- 检测到的文本 (DetectedText)
- 单词与行之间的关系（ Id 和 ParentId ）
- 图像上的文本的位置 (Geometry)
- Amazon Rekognition 对检测到的文本和边界框的准确性的置信度（ Confidence ）
- 检测到的文本的类型 (Type)

检测到的文本

每个 TextDetection 元素均包含在 DetectedText 字段中识别的文本（单词或行）。单词是一个或多个字母字符，不用空格分隔。在一个图像中，DetectText 最多可以检测 100 个单词。返回的文本可能包含使单词无法被识别的字符。例如，用 C@t 替换 Cat。要确定某个 TextDetection 元素是表示一行文本还是一个单词，请使用 Type 字段。

每个 TextDetection 元素均包含一个百分比值，表示 Amazon Rekognition 对检测到的文本和文本周围的边界框的准确性的置信度。

单词与行的关系

每个 TextDetection 元素都有一个标识符字段 Id。Id 显示单词在行中的位置。如果元素是单词，父标识符字段 ParentId 将标识检测到单词的行。行的 ParentId 为 null。例如，示例图中的行“but keep”具有以下 Id 和 ParentId 值：

文本	ID	父级 ID
but keep	3	
but	8	3
keep	9	3

文本在图像上的位置

要确定识别的文本在图像上的位置，请使用 DetectText 返回的边界框 (Geometry) 信息。Geometry 对象包含两种类型的边界框信息，分别适用于检测到的行和单词：

- BoundingBox 对象中轴对齐的粗糙矩形轮廓线
- 点数组中由多个 X 坐标和 Y 坐标组成的更精细的多边形

边界框和多边形坐标将显示文本在源图像上的位置。坐标值是占图像总体尺寸的比率。有关更多信息，请参阅 BoundingBox。

来自 DetectText 操作的以下 JSON 响应显示了下图中检测到的单词和行。

```
{
 'TextDetections': [{'Confidence': 99.35693359375,
                     'DetectedText': "IT'S",
                     'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                                  'Left': 0.6684935688972473,
                                                  'Top': 0.18226495385169983,
                                                  'Width': 0.1461552083492279},
                                  'Polygon': [{'X': 0.6684935688972473,
```

```
                                    'Y': 0.1838926374912262},
                                   {'X': 0.8141663074493408,
                                    'Y': 0.18226495385169983},
                                   {'X': 0.8146487474441528,
                                    'Y': 0.28051772713661194},
                                   {'X': 0.6689760088920593,
                                    'Y': 0.2821454107761383}]},
                  'Id': 0,
                  'Type': 'LINE'},
                 {'Confidence': 99.6207275390625,
                  'DetectedText': 'MONDAY',
                  'Geometry': {'BoundingBox': {'Height': 0.11442459374666214,
                                               'Left': 0.5566731691360474,
                                               'Top': 0.3525116443634033,
                                               'Width': 0.39574965834617615},
                               'Polygon': [{'X': 0.5566731691360474,
                                            'Y': 0.353712260723114},
                                           {'X': 0.9522717595100403,
                                            'Y': 0.3525116443634033},
                                           {'X': 0.9524227976799011,
                                            'Y': 0.4657355844974518},
                                           {'X': 0.5568241477012634,
                                            'Y': 0.46693623065948486}]},
                  'Id': 1,
                  'Type': 'LINE'},
                 {'Confidence': 99.6160888671875,
                  'DetectedText': 'but keep',
                  'Geometry': {'BoundingBox': {'Height': 0.08314694464206696,
                                               'Left': 0.6398131847381592,
                                               'Top': 0.5267938375473022,
                                               'Width': 0.2021435648202896},
                               'Polygon': [{'X': 0.640289306640625,
                                            'Y': 0.5267938375473022},
                                           {'X': 0.8419567942619324,
                                            'Y': 0.5295097827911377},
                                           {'X': 0.8414806723594666,
                                            'Y': 0.609940767288208},
                                           {'X': 0.6398131847381592,
                                            'Y': 0.6072247624397278}]},
                  'Id': 2,
                  'Type': 'LINE'},
                 {'Confidence': 88.95134735107422,
                  'DetectedText': 'Smiling',
                  'Geometry': {'BoundingBox': {'Height': 0.4326171875,
```

```
                                        'Left': 0.46289217472076416,
                                        'Top': 0.5634765625,
                                        'Width': 0.5371078252792358},
                             'Polygon': [{'X': 0.46289217472076416,
                                          'Y': 0.5634765625},
                                         {'X': 1.0, 'Y': 0.5634765625},
                                         {'X': 1.0, 'Y': 0.99609375},
                                         {'X': 0.46289217472076416,
                                          'Y': 0.99609375}]},
                  'Id': 3,
                  'Type': 'LINE'},
                 {'Confidence': 99.35693359375,
                  'DetectedText': "IT'S",
                  'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                               'Left': 0.6684935688972473,
                                               'Top': 0.18226495385169983,
                                               'Width': 0.1461552083492279},
                               'Polygon': [{'X': 0.6684935688972473,
                                            'Y': 0.1838926374912262},
                                           {'X': 0.8141663074493408,
                                            'Y': 0.18226495385169983},
                                           {'X': 0.8146487474441528,
                                            'Y': 0.28051772713661194},
                                           {'X': 0.6689760088920593,
                                            'Y': 0.2821454107761383}]},
                  'Id': 4,
                  'ParentId': 0,
                  'Type': 'WORD'},
                 {'Confidence': 99.6207275390625,
                  'DetectedText': 'MONDAY',
                  'Geometry': {'BoundingBox': {'Height': 0.11442466825246811,
                                               'Left': 0.5566731691360474,
                                               'Top': 0.35251158475875854,
                                               'Width': 0.39574965834617615},
                               'Polygon': [{'X': 0.5566731691360474,
                                            'Y': 0.3537122905254364},
                                           {'X': 0.9522718787193298,
                                            'Y': 0.35251158475875854},
                                           {'X': 0.9524227976799011,
                                            'Y': 0.4657355546951294},
                                           {'X': 0.5568241477012634,
                                            'Y': 0.46693626046180725}]},
                  'Id': 5,
                  'ParentId': 1,
```

```
                    'Type': 'WORD'},
                   {'Confidence': 99.96778869628906,
                    'DetectedText': 'but',
                    'Geometry': {'BoundingBox': {'Height': 0.0625,
                                                 'Left': 0.6402802467346191,
                                                 'Top': 0.5283203125,
                                                 'Width': 0.08027780801057816},
                                 'Polygon': [{'X': 0.6402802467346191,
                                              'Y': 0.5283203125},
                                             {'X': 0.7205580472946167,
                                              'Y': 0.5283203125},
                                             {'X': 0.7205580472946167,
                                              'Y': 0.5908203125},
                                             {'X': 0.6402802467346191,
                                              'Y': 0.5908203125}]},
                    'Id': 6,
                    'ParentId': 2,
                    'Type': 'WORD'},
                   {'Confidence': 99.26438903808594,
                    'DetectedText': 'keep',
                    'Geometry': {'BoundingBox': {'Height': 0.0818721204996109,
                                                 'Left': 0.7344760298728943,
                                                 'Top': 0.5280686020851135,
                                                 'Width': 0.10748066753149033},
                                 'Polygon': [{'X': 0.7349520921707153,
                                              'Y': 0.5280686020851135},
                                             {'X': 0.8419566750526428,
                                              'Y': 0.5295097827911377},
                                             {'X': 0.8414806127548218,
                                              'Y': 0.6099407076835632},
                                             {'X': 0.7344760298728943,
                                              'Y': 0.6084995269775391}]},
                    'Id': 7,
                    'ParentId': 2,
                    'Type': 'WORD'},
                   {'Confidence': 88.95134735107422,
                    'DetectedText': 'Smiling',
                    'Geometry': {'BoundingBox': {'Height': 0.4326171875,
                                                 'Left': 0.46289217472076416,
                                                 'Top': 0.5634765625,
                                                 'Width': 0.5371078252792358},
                                 'Polygon': [{'X': 0.46289217472076416,
                                              'Y': 0.5634765625},
                                             {'X': 1.0, 'Y': 0.5634765625},
```

```
                                                {'X': 1.0, 'Y': 0.99609375},
                                                {'X': 0.46289217472076416,
                                                 'Y': 0.99609375}]},
                    'Id': 8,
                    'ParentId': 3,
                    'Type': 'WORD'}],
 'TextModelVersion': '3.0'}
```

检测存储视频中的文本

存储视频中的 Amazon Rekognition Video 文本检测是一个异步操作。要开始检测文本，请致电StartTextDetection。Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon SNS 主题。如果视频分析成功，请调用 GetTextDetection 来获取分析结果。有关启动视频分析和获取结果的详细信息，请参阅调用 Amazon Rekognition Video 操作。

此过程扩展了使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)中的代码。它使用 Amazon SQS 队列获取视频分析请求的完成状态。

检测存储在 Amazon S3 存储桶内的视频中的文本 (SDK)

1. 按照使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)中的步骤操作。
2. 将以下代码添加到步骤 1 中的 `VideoDetect` 类。

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


private static void StartTextDetection(String bucket, String video) throws
 Exception{

    NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);

    StartTextDetectionRequest req = new StartTextDetectionRequest()
            .withVideo(new Video()
                    .withS3Object(new S3Object()
                        .withBucket(bucket)
```

```
                        .withName(video)))
            .withNotificationChannel(channel);


    StartTextDetectionResult startTextDetectionResult =
 rek.startTextDetection(req);
    startJobId=startTextDetectionResult.getJobId();

}

private static void GetTextDetectionResults() throws Exception{

    int maxResults=10;
    String paginationToken=null;
    GetTextDetectionResult textDetectionResult=null;

    do{
        if (textDetectionResult !=null){
            paginationToken = textDetectionResult.getNextToken();

        }


        textDetectionResult = rek.getTextDetection(new
 GetTextDetectionRequest()
             .withJobId(startJobId)
             .withNextToken(paginationToken)
             .withMaxResults(maxResults));

        VideoMetadata videoMetaData=textDetectionResult.getVideoMetadata();

        System.out.println("Format: " + videoMetaData.getFormat());
        System.out.println("Codec: " + videoMetaData.getCodec());
        System.out.println("Duration: " + videoMetaData.getDurationMillis());
        System.out.println("FrameRate: " + videoMetaData.getFrameRate());


        //Show text, confidence values
        List<TextDetectionResult> textDetections =
 textDetectionResult.getTextDetections();


        for (TextDetectionResult text: textDetections) {
            long seconds=text.getTimestamp()/1000;
```

```
            System.out.println("Sec: " + Long.toString(seconds) + " ");
            TextDetection detectedText=text.getTextDetection();

            System.out.println("Text Detected: " +
 detectedText.getDetectedText());
                System.out.println("Confidence: " +
 detectedText.getConfidence().toString());
                System.out.println("Id : " + detectedText.getId());
                System.out.println("Parent Id: " + detectedText.getParentId());
                System.out.println("Bounding Box" +
 detectedText.getGeometry().getBoundingBox().toString());
                System.out.println("Type: " + detectedText.getType());
                System.out.println();
        }
    } while (textDetectionResult !=null && textDetectionResult.getNextToken() !=
 null);


}
```

在函数 `main` 中，将以下行:

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

替换为:

```
        StartTextDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetTextDetectionResults();
```

Java V2

此代码取自 AWS 文档 SDK 示例 GitHub 存储库。请在此处查看完整示例。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextVideo {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
         "   video - The name of video (for example, people.mp4). \n\n" +
         "   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic. \n\n" +
         "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

     if (args.length != 4) {
```

```
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
public static void startTextLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
```

```
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (textDetectionResponse !=null)
                paginationToken = textDetectionResponse.nextToken();

            GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
```

```
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
 textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
 detectedText.textDetection().id());
                System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                System.out.println("Type: " +
 detectedText.textDetection().type());
                System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                System.out.println();
            }

        } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

Python

```
#Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def StartTextDetection(self):
        response=self.rek.start_text_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetTextDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_text_detection(JobId=self.startJobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print('Codec: ' + response['VideoMetadata']['Codec'])

            print('Duration: ' + str(response['VideoMetadata']
['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for textDetection in response['TextDetections']:
                text=textDetection['TextDetection']

                print("Timestamp: " + str(textDetection['Timestamp']))
                print("   Text Detected: " + text['DetectedText'])
                print("   Confidence: " +  str(text['Confidence']))
                print ("      Bounding box")
                print ("        Top: " + str(text['Geometry']['BoundingBox']
['Top']))
```

```
                print ("        Left: " + str(text['Geometry']['BoundingBox']
['Left']))
                print ("        Width: " +  str(text['Geometry']['BoundingBox']
['Width']))
                print ("        Height: " +  str(text['Geometry']['BoundingBox']
['Height']))
                print ("   Type: " + str(text['Type']) )
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

在函数 main 中，将以下行:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

替换为:

```
    analyzer.StartTextDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetTextDetectionResults()
```

CLI

运行以下 AWS CLI 命令开始检测视频中的文本。

```
 aws rekognition start-text-detection --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}"\
 --notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
 --region region-name --profile profile-name
```

更新以下值：

- 将amzn-s3-demo-bucket和video-name更改为您在步骤 2 中指定的 Amazon S3 存储桶名称和文件名。
- 将region-name更改为您使用的 AWS 区域。

- 将profile-name的值替换为您的开发人员资料的名称。
- 将 topic-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 3 中创建的 Amazon SNS 主题的 ARN。
- 将 role-ARN 更改为您在 配置 Amazon Rekognition Video 的步骤 7 中创建的 IAM 服务角色的 ARN。

如果您在 Windows 设备上访问 CLI，请使用双引号代替单引号，并用反斜杠（即 \）对内部双引号进行转义，以解决可能遇到的任何解析器错误。有关示例，请参阅以下内容：

```
aws rekognition start-text-detection --video \
 "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
 --notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
 --region region-name --profile profile-name
```

运行后续代码示例后，复制返回的 jobID 并将其提供给以下 GetTextDetection 命令以获取结果，将 job-id-number 替换为您之前收到的 jobID：

```
aws rekognition get-text-detection --job-id job-id-number --profile profile-name
```

> **Note**
>
> 如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

3. 运行该代码。在视频中检测到的文本将显示在列表中。

筛选条件

筛选器是可选的请求参数，可供您在调用 StartTextDetection 时使用。通过按文本区域、大小和置信度评分进行筛选，您可以更加灵活地控制文本检测输出。通过使用感兴趣的区域，您可以轻松将文本检测范围限制为相关的区域，例如，图片的倒数第三个区域或者用于显示足球比赛记分板的左上角。单词边界框大小筛选器可用于避免噪点或不相关的小背景文本。最后，通过单词置信度筛选器，您可以移除因模糊或脏污导致的不可靠的结果。

有关筛选值的信息，请参阅 DetectTextFilters。

您可以使用以下筛选器：

- MinConfidence— 设置字词检测的置信度。从结果中排除检测置信度低于此级别的单词。值应介于 0 和 100 之间。
- MinBoundingBoxWidth— 设置单词边界框的最小宽度。将从结果中排除其边界框小于此值的单词。该值是相对于视频帧宽度的。
- MinBoundingBoxHeight— 设置单词边框的最小高度。将从结果中排除其边界框高度小于此值的单词。该值是相对于视频帧高度的。
- RegionsOfInterest— 将检测范围限制在帧的特定区域。这些值是相对于帧尺寸的。对于仅部分位于区域内的对象，响应是不确定的。

GetTextDetection 响应

GetTextDetection 将返回一个数组 (TextDetectionResults)，其中包含有关在视频中检测到的文本的信息。每当在视频中检测到单词或行时，都会存在一个数组元素 TextDetection。数组元素按时间（从视频开始起计时，以毫秒为单位）进行排序。

以下是来自 GetTextDetection 的部分 JSON 响应。在响应中，请注意以下内容：

- 文本信息-TextDetectionResult 数组元素包含有关检测到的文本 (TextDetection) 和视频中检测到文本的时间 (Timestamp) 的信息。
- 分页信息 – 此示例显示一页文本检测信息。您可以在 GetTextDetection 的 MaxResults 输入参数中指定要返回的文本元素数量。如果存在的结果的数量超过了 MaxResults 或结果数超过默认的最大值，则 GetTextDetection 会返回一个令牌 (NextToken)，用于获取下一页的结果。有关更多信息，请参阅 获取 Amazon Rekognition Video 分析结果。
- 视频信息 - 此响应包含有关由 VideoMetadata 返回的每页信息中的视频格式 (GetTextDetection) 的信息。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 174441,
        "Format": "QuickTime / MOV",
```

```
        "FrameRate": 29.970029830932617,
        "FrameHeight": 480,
        "FrameWidth": 854
    },
    "TextDetections": [
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle Twinkle Little Star",
                "Type": "LINE",
                "Id": 0,
                "Confidence": 99.91780090332031,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.8337579369544983,
                        "Height": 0.08365312218666077,
                        "Left": 0.08313830941915512,
                        "Top": 0.4663468301296234
                    },
                    "Polygon": [
                        {
                            "X": 0.08313830941915512,
                            "Y": 0.4663468301296234
                        },
                        {
                            "X": 0.9168962240219116,
                            "Y": 0.4674469828605652
                        },
                        {
                            "X": 0.916861355304718,
                            "Y": 0.5511001348495483
                        },
                        {
                            "X": 0.08310343325138092,
                            "Y": 0.5499999523162842
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
```

```
                "Type": "WORD",
                "Id": 1,
                "ParentId": 0,
                "Confidence": 99.98338317871094,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.0833333358168602,
                        "Left": 0.08313817530870438,
                        "Top": 0.46666666865348816
                    },
                    "Polygon": [
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
                "Type": "WORD",
                "Id": 2,
                "ParentId": 0,
                "Confidence": 99.982666015625,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.08124999701976776,
```

```
                    "Left": 0.3454332649707794,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.3454332649707794,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.5878220200538635,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.5878220200538635,
                        "Y": 0.550000011920929
                    },
                    {
                        "X": 0.3454332649707794,
                        "Y": 0.550000011920929
                    }
                ]
            }
        }
    },
    {
        "Timestamp": 967,
        "TextDetection": {
            "DetectedText": "Little",
            "Type": "WORD",
            "Id": 3,
            "ParentId": 0,
            "Confidence": 99.8787612915039,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.16627635061740875,
                    "Height": 0.08124999701976776,
                    "Left": 0.6053864359855652,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.6053864359855652,
                        "Y": 0.46875
                    },
```

```
                    {
                        "X": 0.7716627717018127,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.7716627717018127,
                        "Y": 0.550000011920929
                    },
                    {
                        "X": 0.6053864359855652,
                        "Y": 0.550000011920929
                    }
                ]
            }
        }
    },
    {
        "Timestamp": 967,
        "TextDetection": {
            "DetectedText": "Star",
            "Type": "WORD",
            "Id": 4,
            "ParentId": 0,
            "Confidence": 99.82640075683594,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.12997658550739288,
                    "Height": 0.08124999701976776,
                    "Left": 0.7868852615356445,
                    "Top": 0.46875
                },
                "Polygon": [
                    {
                        "X": 0.7868852615356445,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.9168618321418762,
                        "Y": 0.46875
                    },
                    {
                        "X": 0.9168618321418762,
                        "Y": 0.550000011920929
                    },
```

```
                        {
                            "X": 0.7868852615356445,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        }
    ],
    "NextToken": "NiHpGbZFnkM/S8kLcukMni15wb05iKtquu/Mwc+Qg1LVlMjjKNOD0Z0GusSPg7TONLe
+OZ3P",
    "TextModelVersion": "3.0"
}
```

检测存储视频中的视频分段

Amazon Rekognition Video 提供了一个 API，用于识别视频的有用分段，例如黑帧和片尾字幕。

现在，观众观看的内容远超从前。特别是， Over-The-Top（OTT）和 Video-On-Demand（VOD）平台可随时随地在任何屏幕上提供丰富的内容选择。随着内容量的急速增长，媒体公司面临着制作和管理其内容的挑战。这对于提供高质量的观看体验和更好的盈利内容至关重要。如今，公司使用大量经过训练的人力资源来执行下列任务。

- 查找片头和片尾字幕在一段内容中的位置
- 选择合适的位置来插入广告，例如在无声的黑帧序列中
- 将视频分解为较小的剪辑以更好地编制索引

这些手动过程费用昂贵、速度缓慢、无法扩展，不能满足每天制作、授予许可以及从存档中检索内容量需求。

您可以使用 Amazon Rekognition Video，使用由机器学习 (ML) 支持的完全托管、专门构建的视频片段 APIs 检测来自动执行媒体分析任务。通过使用 Amazon Rekognition Video APIs 片段，您可以轻松分析大量视频并检测黑框或镜头变化等标记。在每次检测时，您可以获取 SMPTE（美国电影电视工程师协会）时间码、时间戳和帧数。无需机器学习经验。

Amazon Rekognition Video 会分析存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中的视频。返回的 SMPTE 时间码是精确到帧 — Amazon Rekognition Video 提供检测到的视频分段的确切帧数，并自动处理各种视频帧率格式。您可以使用来自 Amazon Rekognition Video 的精确到帧的元数据自动完成特定任务，从而完全或大幅减少受培训人工操作员的审核工作量，让他们专注于更具创造力的工作。您可以执行编制内容、插入广告等任务，以及在云中大规模添加“狂欢标记”到内容中。

有关定价的信息，请参阅 Amazon Rekognition 定价。

Amazon Rekognition Video 分段检测支持两种类型的分段任务 — 技术提示 检测和镜头检测。

主题

- 技术提示
- 镜头检测
- 关于 Amazon Rekognition Video 分段检测 API
- 使用 Amazon Rekognition 分段 API

- 示例：检测存储视频中的分段

技术提示

技术提示可以识别视频中的黑帧、彩条、片头字幕、片尾字幕、工作室徽标和主要节目内容。

黑帧

视频通常包含不含音频的黑帧，用作插入广告或界定节目段结尾（例如场景或开场字幕）。借助 Amazon Rekognition Video，您可以检测到黑帧序列，以自动插入广告、为 VOD 打包内容以及界定各种节目段或场景。带有音频的黑色帧（例如淡出或画外音）被视为内容且不会返回。

服务抵扣金额

Amazon Rekognition Video 可自动识别电影或电视节目的开头字幕和片尾字幕的开头和结尾的确切帧。利用这些信息，您可以在视频点播 (VOD) 应用程序中生成“狂欢标记”或交互式观看者提示，例如“下一集”或“跳过简介”。您还可以检测视频中节目内容的第一帧和最后一帧。Amazon Rekognition Video 经过培训，可以处理各种片头和片尾字幕风格，从简单的滚动字幕到更具挑战性的字幕以及内容。

彩条

Amazon Rekognition Video 允许您检测显示 SMPTE 彩条的视频部分，SMPTE 彩条是以特定模式显示的一组颜色，可确保在广播监视器、节目和相机上正确校准颜色。有关 SMPTE 彩条的更多信息，请参阅 SMPTE 彩条。当彩条连续显示为默认信号而不是内容时，此元数据可用于通过从内容中删除彩条段来为 VOD 应用程序准备内容，也可用于检测录音中的广播信号丢失等问题。

画面

画面是视频中通常接近开头的部分，其中包含有关剧集、工作室、视频格式、音频频道等的文本元数据。Amazon Rekognition Video 可以识别画面的开头和结尾，从而在准备内容以供最终观看时轻松使用文本元数据或移除画面。

工作室徽标

工作室徽标是显示参与制作该节目的制作工作室徽标或标志的序列。Amazon Rekognition Video 可以检测到这些序列，以便用户可以查看它们以识别工作室。

内容

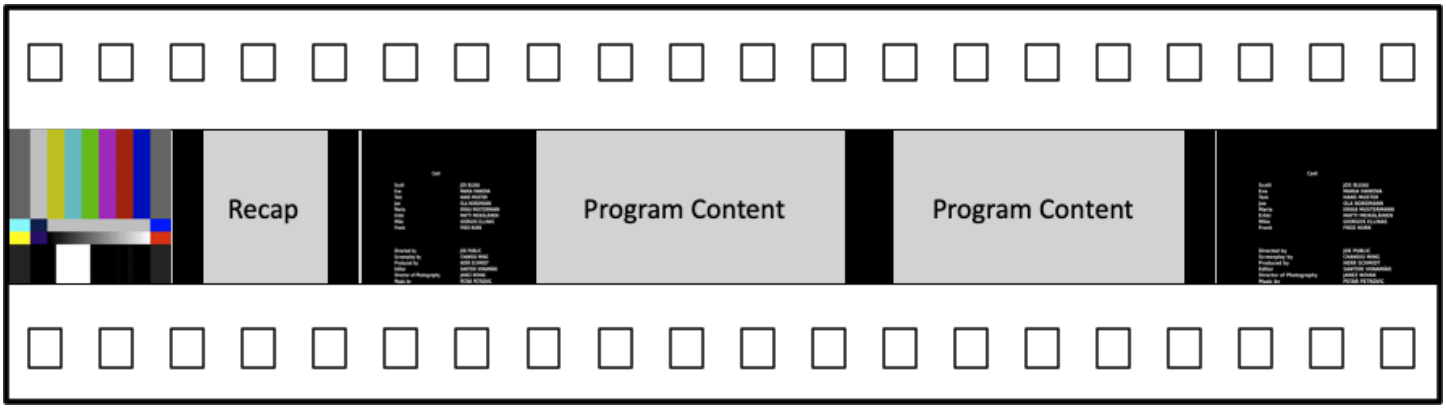
内容是电视节目或电影中包含节目或相关元素的部分。黑帧、字幕、彩条、画面和工作室徽标不被视为内容。Amazon Rekognition Video 可以检测视频中每个内容分段的开始和结尾，因此您可以找到节目的运行时间或特定分段。

内容分段包括但不限于以下内容：

- 在两个广告时段之间的节目场景
- 在视频开头对上一集进行的快速回顾
- 字幕后的彩蛋内容
- “无文本”内容，例如一组所有节目场景，这些场景最初包含叠加的文本，但为了支持翻译成其他语言，文本已被删除。

在 Amazon Rekognition Video 完成对所有内容分段的检测后，您可以应用领域知识或将其发送给人工审核，以进一步对每个分段进行分类。例如，如果您使用的视频总是以回顾开头，则可以将第一个内容分段归类为回顾。

下图说明了节目或影片时间线上的技术提示分段。请注意彩条和开场字幕、内容分段（例如回顾和主节目）、整个视频中的黑帧以及片尾字幕。



镜头检测

镜头是由一台相机连续拍摄的一系列相互关联的连续照片，表示在时间和空间上的连续动作。借助 Amazon Rekognition Video，您可以检测每个镜头的开始、结束和持续时间，并且可以计数内容分段中的所有镜头。您可以将镜头元数据用于以下任务。

- 使用选定的镜头创建宣传视频。

- 在不影响观众体验的位置插入广告，例如镜头中间有人说话时。
- 生成一组预览缩略图，避开镜头之间的过渡内容。

镜头检测在硬切换到另一个镜头的确切帧上进行标记。如果存在从一个镜头向另一个镜头的软过渡，则 Amazon Rekognition Video 忽略过渡。这可确保镜头的开始和结束时间不包括没有实际内容的部分。

下图说明了一段胶片上的镜头检测段。请注意，每个镜头都是由从一个镜头角度或位置到下一个镜头的切换来识别的。

关于 Amazon Rekognition Video 分段检测 API

要对存储的视频进行分段，您可以使用异步StartSegmentDetection和 GetSegmentDetectionAPI 操作来启动分段作业并获取结果。分段检测接受存储在 Amazon S3 存储桶中的视频并返回 JSON 输出。通过配置 `StartSegmentdetection` API 请求，您可以选择仅检测技术提示、仅检测镜头变化或检测技术提示和镜头变化。您还可以为最低预测置信度设置阈值来筛选检测到的分段。有关更多信息，请参阅 使用 Amazon Rekognition 分段 API 。有关代码示例，请参阅 示例：检测存储视频中的分段。

使用 Amazon Rekognition 分段 API

存储视频中 Amazon Rekognition Video 分段检测是一项 Amazon Rekognition Video 异步操作。Amazon Rekognition 分段 API 是一种复合 API，您可以从单个 API 调用中选择分析类型（技术提示或镜头检测）。有关调用异步操作的信息，请参阅调用 Amazon Rekognition Video 操作。

主题

- 起始段分析
- 获取分段分析结果

起始段分析

要开始检测存储视频中的分段，请调用 StartSegmentDetection。输入参数与其他 Amazon Rekognition Video 操作相同，但添加了分段类型选择和结果筛选。有关更多信息，请参阅 启动视频分析。

以下是 StartSegmentDetection 传递的 JSON 示例。请求同时指定检测技术提示和镜头检测分段。为技术提示分段 (90%) 和镜头检测分段 (80%) 请求了不同的筛选器，以实现最低检测置信度。

```
{
  "Video": {
    "S3Object": {
      "Bucket": "test_files",
      "Name": "test_file.mp4"
    }
    "SegmentTypes":["TECHNICAL_CUES", "SHOT"]
    "Filters": {
      "TechnicalCueFilter": {
         "MinSegmentConfidence": 90,
         "BlackFrame" : {
            "MaxPixelThreshold": 0.1,
            "MinCoveragePercentage": 95
         }
      },
      "ShotFilter" : {
          "MinSegmentConfidence": 60
      }
  }
}
```

选择分段类型

使用 SegmentTypes 数组输入参数检测输入视频中的技术提示和/或镜头检测分段。

- 技术提示 — 识别在视频中检测到的技术提示（黑帧、彩条、片头字幕、片尾字幕、工作室徽标和主要节目内容）的开始、结束和持续时间的精确到帧的时间戳。例如，您可以使用技术提示来查找片尾字幕的开始。有关更多信息，请参阅 技术提示。
- 镜头 — 识别镜头的开始、结束和持续时间。例如，您可以使用镜头检测来确定一个视频的最终编辑候选镜头。有关更多信息，请参阅 镜头检测。

筛选分析结果

您可以使用 Filters (StartSegmentDetectionFilters) 输入参数指定响应中返回的最低检测可信度。在 Filters 中，使用 ShotFilter (StartShotDetectionFilter) 筛选检测到的镜头。使用 TechnicalCueFilter (StartTechnicalCueDetectionFilter) 筛选技术线索。

有关代码示例，请参阅 示例：检测存储视频中的分段。

获取分段分析结果

Amazon Rekognition Video 会将视频分析的完成状态发布到 Amazon Simple Notification Service 主题。如果视频分析成功，请致电GetSegmentDetection以获取视频分析的结果。

以下是一个示例 GetSegmentDetection 请求。JobId 是从对 StartSegmentDetection 的调用中返回的作业标识符。有关其他输入参数的信息，请参阅获取 Amazon Rekognition Video 分析结果。

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw=="
}
```

GetSegmentDetection 返回所请求分析的结果以及有关存储的视频的常规信息。

一般信息

GetSegmentDection 返回以下常规信息。

- 音频信息-响应包括AudioMetadata对象数组中的音频元数据。AudioMetadata可以有多个音频流。每个 AudioMetadata 对象都包含一个音频流的元数据。AudioMetadata 对象中的音频信息包括音频编解码器、音频通道数量、音频流的持续时间和采样率。音频元数据在 GetSegmentDetection 返回的每个信息页面中返回。
- 视频信息 — 目前，Amazon Rekognition Video 返回数组中的单个对象。VideoMetadataVideoMetadata该对象包含有关 Amazon Rekognition Video 选择分析的输入文件中视频流的信息。VideoMetadata 对象包括视频编解码器、视频格式和其他信息。视频元数据在 GetSegmentDetection 返回的每个信息页面中返回。
- 分页信息 – 示例显示了分段信息的一页。您可以在 GetSegmentDetection 的 MaxResults 输入参数中指定要返回的元素数量。如果存在的结果的数量超过了 MaxResults，则

GetSegmentDetection 会返回一个令牌 (NextToken)，用于获取下一页的结果。有关更多信息，请参阅 获取 Amazon Rekognition Video 分析结果。

- 请求信息 – 在对 StartSegmentDetection 的调用中请求的分析类型在 SelectedSegmentTypes 字段中返回。

Segments

在视频中检测到的技术提示和镜头信息在 SegmentDetection 对象的数组 Segments 中返回。输入按照在 StartSegmentDetection 的 SegmentTypes 输入参数中指定的分段类型（TECHNICAL_CUE 或 SHOT）排序。在每个分段类型中，数组按时间戳值排序。每个 SegmentDetection 对象都包括有关所检测到分段类型的信息（技术提示或镜头检测）以及常规信息，例如分段的开始时间、结束时间和持续时间。

时间信息以三种格式返回。

- 毫秒

  自视频开始以来的毫秒数。字段 DurationMillis、StartTimestampMillis 和 EndTimestampMillis 采用毫秒格式。

- 时间码

  Amazon Rekognition Video 时间码采用 SMPTE 格式，视频的每个帧都有一个唯一的时间码值。格式为 hh:mm:ss:frame。例如，时间码值 01:05:40:07 将被读取为 1 小时 5 分 47 秒第 7 帧。Amazon Rekognition Video 支持丢帧率用例。丢帧率时间码格式为 hh:mm:ss;frame。字段 DurationSMPTE、StartTimecodeSMPTE 和 EndTimecodeSMPTE 采用时间码格式。

- 帧计数器

  每个视频分段的持续时间也用帧数表示。该字段 StartFrameNumber 给出视频分段开头的帧数，EndFrameNumber 给出视频分段末尾的帧数。DurationFrames 给出视频分段的总帧数。这些值是使用以 0 开头的帧索引计算的。

您可以使用 SegmentType 字段确定 Amazon Rekognition Video 返回的分段的类型。

- 技术线索 — 该TechnicalCueSegment字段是一个包含探测置信度和技术线索类型的TechnicalCueSegment对象。技术提示的类型有 ColorBars、EndCredits、BlackFrames、OpeningCredits、StudioLogo、Slate和Content。

- 镜头 — 该ShotSegment字段是一个包含检测置信度和视频中镜头片段标识符的ShotSegment对象。

以下示例是 GetSegmentDetection 的 JSON 响应。


```
{
    "SelectedSegmentTypes": [
        {
            "ModelVersion": "2.0",
            "Type": "SHOT"
        },
        {
            "ModelVersion": "2.0",
            "Type": "TECHNICAL_CUE"
        }
    ],
    "Segments": [
        {
            "DurationFrames": 299,
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "TECHNICAL_CUE",
            "TechnicalCueSegment": {
                "Confidence": 90.45006561279297,
                "Type": "BlackFrames"
            }
        },
        {
            "DurationFrames": 150,
            "DurationSMPTE": "00:00:05;00",
            "StartFrameNumber": 299,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 9976,
            "DurationMillis": 5005,
            "StartTimecodeSMPTE": "00:00:09;29",
```

```
            "Type": "TECHNICAL_CUE",
            "TechnicalCueSegment": {
                "Confidence": 100.0,
                "Type": "Content"
            }
        },
        {
            "DurationFrames": 299,
            "ShotSegment": {
                "Index": 0,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "SHOT"
        },
        {
            "DurationFrames": 149,
            "ShotSegment": {
                "Index": 1,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:04;29",
            "StartFrameNumber": 300,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 10010,
            "DurationMillis": 4971,
            "StartTimecodeSMPTE": "00:00:10;00",
            "Type": "SHOT"
        }
    ],
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": [
        {
            "Format": "QuickTime / MOV",
            "FrameRate": 29.970029830932617,
```

```
            "Codec": "h264",
            "DurationMillis": 15015,
            "FrameHeight": 1080,
            "FrameWidth": 1920,
            "ColorRange": "LIMITED"

        }
    ],
    "AudioMetadata": [
        {
            "NumberOfChannels": 1,
            "SampleRate": 48000,
            "Codec": "aac",
            "DurationMillis": 15007
        }
    ]
}
```

有关代码示例，请参阅 示例：检测存储视频中的分段。

示例：检测存储视频中的分段

以下过程说明如何针对存储在 Amazon S3 存储桶中的视频，检测技术提示分段和镜头检测分段。此过程还显示了如何根据 Amazon Rekognition Video 对检测准确性的置信度筛选检测到的分段。

此示例扩展了 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)（使用 Amazon Simple Queue Service 队列获取视频分析请求的完成状态）中的代码。

检测存储在 Amazon S3 存储桶的视频中的分段 (SDK)

1. 执行使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK)。
2. 将以下内容添加到您在步骤 1 中使用的代码。

   Java

   1. 添加以下导入。

```
import com.amazonaws.services.rekognition.model.GetSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.GetSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.SegmentDetection;
import com.amazonaws.services.rekognition.model.SegmentType;
import com.amazonaws.services.rekognition.model.SegmentTypeInfo;
```

```
import com.amazonaws.services.rekognition.model.ShotSegment;
import
 com.amazonaws.services.rekognition.model.StartSegmentDetectionFilters;
import
 com.amazonaws.services.rekognition.model.StartSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.StartSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.StartShotDetectionFilter;
import
 com.amazonaws.services.rekognition.model.StartTechnicalCueDetectionFilter;
import com.amazonaws.services.rekognition.model.TechnicalCueSegment;
import com.amazonaws.services.rekognition.model.AudioMetadata;
```

2. 将以下代码添加类 VideoDetect。

```
   //Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights
 Reserved.
   //PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


   private static void StartSegmentDetection(String bucket, String video)
 throws Exception{

       NotificationChannel channel= new NotificationChannel()
               .withSNSTopicArn(snsTopicArn)
               .withRoleArn(roleArn);

       float minTechnicalCueConfidence = 80F;
       float minShotConfidence = 80F;

       StartSegmentDetectionRequest req = new
 StartSegmentDetectionRequest()
               .withVideo(new Video()
                       .withS3Object(new S3Object()
                               .withBucket(bucket)
                               .withName(video)))
               .withSegmentTypes("TECHNICAL_CUE" , "SHOT")
               .withFilters(new StartSegmentDetectionFilters()
                       .withTechnicalCueFilter(new
 StartTechnicalCueDetectionFilter()

 .withMinSegmentConfidence(minTechnicalCueConfidence))
                       .withShotFilter(new StartShotDetectionFilter()
```

```
 .withMinSegmentConfidence(minShotConfidence)))
                .withJobTag("DetectingVideoSegments")
                .withNotificationChannel(channel);

        StartSegmentDetectionResult startLabelDetectionResult =
 rek.startSegmentDetection(req);
        startJobId=startLabelDetectionResult.getJobId();

    }

    private static void GetSegmentDetectionResults() throws Exception{

        int maxResults=10;
        String paginationToken=null;
        GetSegmentDetectionResult segmentDetectionResult=null;
        Boolean firstTime=true;


        do {
            if (segmentDetectionResult !=null){
                paginationToken = segmentDetectionResult.getNextToken();
            }

            GetSegmentDetectionRequest segmentDetectionRequest= new
 GetSegmentDetectionRequest()
                    .withJobId(startJobId)
                    .withMaxResults(maxResults)
                    .withNextToken(paginationToken);

            segmentDetectionResult =
 rek.getSegmentDetection(segmentDetectionRequest);

            if(firstTime) {
                System.out.println("\nStatus\n------");
                System.out.println(segmentDetectionResult.getJobStatus());
                System.out.println("\nRequested features
\n------------------");
                 for (SegmentTypeInfo requestedFeatures :
 segmentDetectionResult.getSelectedSegmentTypes()) {
                    System.out.println(requestedFeatures.getType());
                }
                 int count=1;
```

```
                List<VideoMetadata> videoMetaDataList =
segmentDetectionResult.getVideoMetadata();
                System.out.println("\nVideo Streams\n-------------");
                for (VideoMetadata videoMetaData: videoMetaDataList) {
                    System.out.println("Stream: " + count++);
                    System.out.println("\tFormat: " +
videoMetaData.getFormat());
                    System.out.println("\tCodec: " +
videoMetaData.getCodec());
                    System.out.println("\tDuration: " +
videoMetaData.getDurationMillis());
                    System.out.println("\tFrameRate: " +
videoMetaData.getFrameRate());
                }


                List<AudioMetadata> audioMetaDataList =
segmentDetectionResult.getAudioMetadata();
                System.out.println("\nAudio streams\n-------------");

                count=1;
                for (AudioMetadata audioMetaData: audioMetaDataList) {
                    System.out.println("Stream: " + count++);
                    System.out.println("\tSample Rate: " +
audioMetaData.getSampleRate());
                    System.out.println("\tCodec: " +
audioMetaData.getCodec());
                    System.out.println("\tDuration: " +
audioMetaData.getDurationMillis());
                    System.out.println("\tNumber of Channels: " +
audioMetaData.getNumberOfChannels());
                }
                System.out.println("\nSegments\n--------");

                firstTime=false;
            }


            //Show segment information

            List<SegmentDetection> detectedSegments=
segmentDetectionResult.getSegments();

            for (SegmentDetection detectedSegment: detectedSegments) {
```

```
            if
(detectedSegment.getType().contains(SegmentType.TECHNICAL_CUE.toString()))
{
                    System.out.println("Technical Cue");
                    TechnicalCueSegment
segmentCue=detectedSegment.getTechnicalCueSegment();
                    System.out.println("\tType: " + segmentCue.getType());
                    System.out.println("\tConfidence: " +
segmentCue.getConfidence().toString());
                }
            if
(detectedSegment.getType().contains(SegmentType.SHOT.toString())) {
                    System.out.println("Shot");
                    ShotSegment
segmentShot=detectedSegment.getShotSegment();
                    System.out.println("\tIndex " +
segmentShot.getIndex());
                    System.out.println("\tConfidence: " +
segmentShot.getConfidence().toString());
                }
                long seconds=detectedSegment.getDurationMillis();
                System.out.println("\tDuration : " + Long.toString(seconds)
+ " milliseconds");
                System.out.println("\tStart time code: " +
detectedSegment.getStartTimecodeSMPTE());
                System.out.println("\tEnd time code: " +
detectedSegment.getEndTimecodeSMPTE());
                System.out.println("\tDuration time code: " +
detectedSegment.getDurationSMPTE());
                System.out.println();

            }

       } while (segmentDetectionResult !=null &&
segmentDetectionResult.getNextToken() != null);

   }
```

3. 在函数 main 中，将以下行:

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

       if (GetSQSMessageSuccess()==true)
```

```
         GetLabelDetectionResults();
```

替换为:

```
        StartSegmentDetection(amzn-s3-demo-bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetSegmentDetectionResults();
```

Java V2

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectVideoSegments {
```

```
private static String startJobId ="";
public static void main(String[] args) {

    final String usage = "\n" +
        "Usage: " +
        "   <bucket> <video> <topicArn> <roleArn>\n\n" +
        "Where:\n" +
        "   bucket - The name of the bucket in which the video is located (for
example, (for example, amzn-s3-demo-bucket). \n\n"+
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_WEST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
```

```
public static void startTextLabels(RekognitionClient rekClient,
                                NotificationChannel channel,
                                String bucket,
                                String video) {
    try {
        S3Object s3Obj = S3Object.builder()
            .bucket(bucket)
            .name(video)
            .build();

        Video vidOb = Video.builder()
            .s3Object(s3Obj)
            .build();

        StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
            .jobTag("DetectingLabels")
            .notificationChannel(channel)
            .video(vidOb)
            .build();

        StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
        startJobId = labelDetectionResponse.jobId();

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}

public static void GetTextResults(RekognitionClient rekClient) {

    try {
        String paginationToken=null;
        GetTextDetectionResponse textDetectionResponse=null;
        boolean finished = false;
        String status;
        int yy=0 ;

        do{
            if (textDetectionResponse !=null)
                paginationToken = textDetectionResponse.nextToken();
```

```
            GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                .jobId(startJobId)
                .nextToken(paginationToken)
                .maxResults(10)
                .build();

            // Wait until the job succeeds.
            while (!finished) {
                textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                status = textDetectionResponse.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null.
            VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<TextDetectionResult> labels=
textDetectionResponse.textDetections();
            for (TextDetectionResult detectedText: labels) {
                System.out.println("Confidence: " +
detectedText.textDetection().confidence().toString());
                System.out.println("Id : " +
detectedText.textDetection().id());
                System.out.println("Parent Id: " +
detectedText.textDetection().parentId());
                System.out.println("Type: " +
detectedText.textDetection().type());
```

```
                System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                System.out.println();
            }

        } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

Python

1. 将以下代码添加到您在步骤 1 中创建的类 VideoDetect。

```
# Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def StartSegmentDetection(self):

        min_Technical_Cue_Confidence = 80.0
        min_Shot_Confidence = 80.0
        max_pixel_threshold = 0.1
        min_coverage_percentage = 60

        response = self.rek.start_segment_detection(
            Video={"S3Object": {"Bucket": self.bucket, "Name": self.video}},
            NotificationChannel={
                "RoleArn": self.roleArn,
                "SNSTopicArn": self.snsTopicArn,
            },
            SegmentTypes=["TECHNICAL_CUE", "SHOT"],
            Filters={
                "TechnicalCueFilter": {
                    "BlackFrame": {
                        "MaxPixelThreshold": max_pixel_threshold,
                        "MinCoveragePercentage": min_coverage_percentage,
```

```
                    },
                    "MinSegmentConfidence": min_Technical_Cue_Confidence,
                },
                "ShotFilter": {"MinSegmentConfidence": min_Shot_Confidence},
            }
        )

        self.startJobId = response["JobId"]
        print(f"Start Job Id: {self.startJobId}")

    def GetSegmentDetectionResults(self):
        maxResults = 10
        paginationToken = ""
        finished = False
        firstTime = True

        while finished == False:
            response = self.rek.get_segment_detection(
                JobId=self.startJobId, MaxResults=maxResults,
NextToken=paginationToken
            )

            if firstTime == True:
                print(f"Status\n------\n{response['JobStatus']}")
                print("\nRequested Types\n---------------")
                for selectedSegmentType in response['SelectedSegmentTypes']:
                    print(f"\tType: {selectedSegmentType['Type']}")
                    print(f"\t\tModel Version:
{selectedSegmentType['ModelVersion']}")

                print()
                print("\nAudio metadata\n--------------")
                for audioMetadata in response['AudioMetadata']:
                    print(f"\tCodec: {audioMetadata['Codec']}")
                    print(f"\tDuration: {audioMetadata['DurationMillis']}")
                    print(f"\tNumber of Channels:
{audioMetadata['NumberOfChannels']}")
                    print(f"\tSample rate: {audioMetadata['SampleRate']}")
                print()
                print("\nVideo metadata\n--------------")
                for videoMetadata in response["VideoMetadata"]:
                    print(f"\tCodec: {videoMetadata['Codec']}")
                    print(f"\tColor Range: {videoMetadata['ColorRange']}")
                    print(f"\tDuration: {videoMetadata['DurationMillis']}")
```

```
                    print(f"\tFormat: {videoMetadata['Format']}")
                    print(f"\tFrame rate: {videoMetadata['FrameRate']}")
                    print("\nSegments\n--------")

                firstTime = False

            for segment in response['Segments']:

                if segment["Type"] == "TECHNICAL_CUE":
                    print("Technical Cue")
                    print(f"\tConfidence: {segment['TechnicalCueSegment']
['Confidence']}")
                    print(f"\tType: {segment['TechnicalCueSegment']
['Type']}")

                if segment["Type"] == "SHOT":
                    print("Shot")
                    print(f"\tConfidence: {segment['ShotSegment']
['Confidence']}")
                    print(f"\tIndex: " + str(segment["ShotSegment"]
["Index"]))

                print(f"\tDuration (milliseconds):
 {segment['DurationMillis']}")
                print(f"\tStart Timestamp (milliseconds):
 {segment['StartTimestampMillis']}")
                print(f"\tEnd Timestamp (milliseconds):
 {segment['EndTimestampMillis']}")

                print(f"\tStart timecode: {segment['StartTimecodeSMPTE']}")
                print(f"\tEnd timecode: {segment['EndTimecodeSMPTE']}")
                print(f"\tDuration timecode: {segment['DurationSMPTE']}")

                print(f"\tStart frame number {segment['StartFrameNumber']}")
                print(f"\tEnd frame number: {segment['EndFrameNumber']}")
                print(f"\tDuration frames: {segment['DurationFrames']}")

                print()

            if "NextToken" in response:
                paginationToken = response["NextToken"]
            else:
                finished = True
```

2. 在函数 `main` 中，将以下行:

```
analyzer.StartLabelDetection()
if analyzer.GetSQSMessageSuccess()==True:
    analyzer.GetLabelDetectionResults()
```

替换为:

```
analyzer.StartSegmentDetection()
if analyzer.GetSQSMessageSuccess()==True:
    analyzer.GetSegmentDetectionResults()
```

> **Note**
>
> 如果您已经运行了除 使用 Java 或 Python 分析存储在 Amazon S3 存储桶中的视频 (SDK) 之外的视频示例，则要替换的代码可能会有所不同。

3. 运行该代码。将显示在输入视频中检测到的分段的相关信息。

检测人脸活跃度

Amazon Rekognition Face Liveness 可以帮助您验证正在进行面部验证的用户是否真实出现在摄像头前。它可以检测出现在摄像头前的仿冒攻击或绕过摄像头的攻击。用户可以通过自拍短视频来完成 Face Liveness 检查，其中，自拍短视频使其根据一系列提示来验证自己是否真实存在。

Face Liveness 通过概率计算确定，然后在检查后返回置信度分数（介于 0—100 之间）。分数越高，对于接受检查的人是真实的就越有信心。Face Liveness 还会返回一个框架，称为参考图像，可用于人脸比较和搜索。与任何基于概率的系统一样，Face Liveness 不能保证完美的结果。将其与其他因素结合使用，对用户的个人身份做出基于风险的决定。

Face Liveness 使用多个组件：

- AWS 带组件的 Amplify SDK（React、Swift (iOS) 和安卓） FaceLivenessDetector
- AWS SDKs
- AWS 云 APIs

当您将应用程序配置为与 Face Liveness 功能集成时，它会使用以下 API 操作：

- CreateFaceLivenessSession-启动 Face Liveness 会话，让您的应用程序中使用 Face Liveness 检测模型。 SessionId 为创建的会话返回 a。还允许您设置自己的 ChallengePrefrence，因此您可以使用该 FaceMovementChallenge 选项。
- StartFaceLivenessSession-由 AWS Ampli FaceLivenessDetector fy 调用。启动一个事件流，其中包含当前会话中相关事件和属性的信息。
- GetFaceLivenessSessionResults-检索特定 Face Liveness 会话的结果，包括 Face Liveness 置信度分数、参考图像和审核图像。

您将使用 AWS Amplify SDK 将 Face Liveness 功能与网络应用程序的基于人脸的验证工作流程集成。当用户通过您的应用程序加入或进行身份验证时，请将其发送到 Amplify SDK 中的 Face Liveness 检查工作流程。Amplify SDK 可在用户拍摄视频自拍时处理用户界面和实时反馈。

当使用 FaceMovementAndLightChallenge 用户的脸部移动到设备上显示的椭圆形时，Amplify SDK 会在屏幕上显示一系列彩色灯光。然后，它会将自拍视频安全地传输到云端 APIs。或者，在使用时 FaceMovementChallenge，用户的脸会移动到设备上显示的椭圆形中，但没有彩色灯光序列。虽然 “FaceMovementAndLightChallenge” 仍然是最大限度地提高准确性的最佳设置，但 FaceMovementChallenge “” 允许客户优先考虑更快的活体检查速度而不是准确性。在这些设置之间进

行选择时，客户应考虑其用例要求，包括预期的攻击类型、所需的错误接受率和错误拒绝率，以及其他检查，例如地理位置（例如基于 IP）、一次性通行码 (OTPs) 等。客户应在针对其特定内容使用不同的置信度分数阈值测试直播效果后做出此决定。此外，对于这两种直播检查类型，应用程序所有者都应实施控制措施来保护发送视频流的设备。分析完成后，您将在后端收到以下信息：

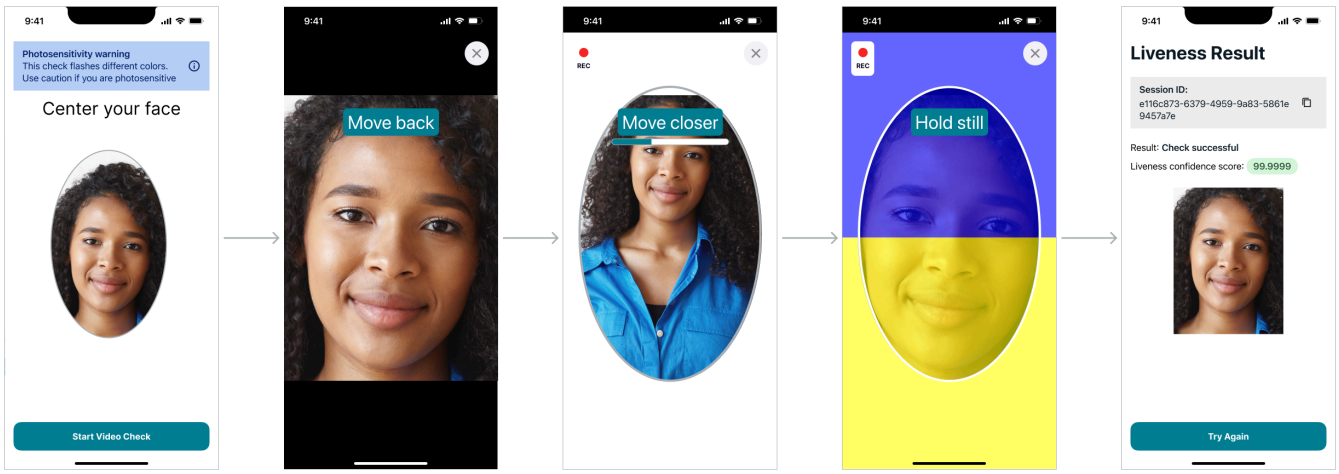
- Face Liveness 置信度分数（介于 0 到 100 之间）
- 一种名为参考图像的高质量图像，可用于人脸匹配或人脸搜索
- 从自拍视频中选择一组图像（最多四张），称为审核图像

Face Liveness 可以用于各种用例。例如，Face Liveness可以与人脸匹配（使用CompareFaces和SearchFacesByImage）一起用于身份验证，在基于年龄的访问限制的平台上进行年龄估算，以及在威慑机器人的同时检测真实的人类用户。

您可以在 Rekognition Face Liveness AI 服务卡中详细了解该服务适用于哪些使用案例、该服务如何使用机器学习（ML），以及负责任地设计和使用该服务时的关键注意事项。

您可以为 Face Liveness 和人脸匹配置信度分数设置阈值。您选择的阈值应反映您的使用案例。然后，您根据分数高于或低于阈值向用户发送身份验证 approval/denial 。如果被拒绝，请用户重试或将其发送到其他方法。

下图演示了从指令到活跃度检查再到返回结果的用户流程：



用户端面部活跃度要求

Amazon Rekognition Face Liveness 要求满足以下最低规格：

设备：

- 设备必须有前置摄像头
- 设备显示器的最低刷新率：60 Hz
- 最小显示器或屏幕尺寸：4 英寸
- 设备不应越狱或被获取 root 权限

摄像头规格：

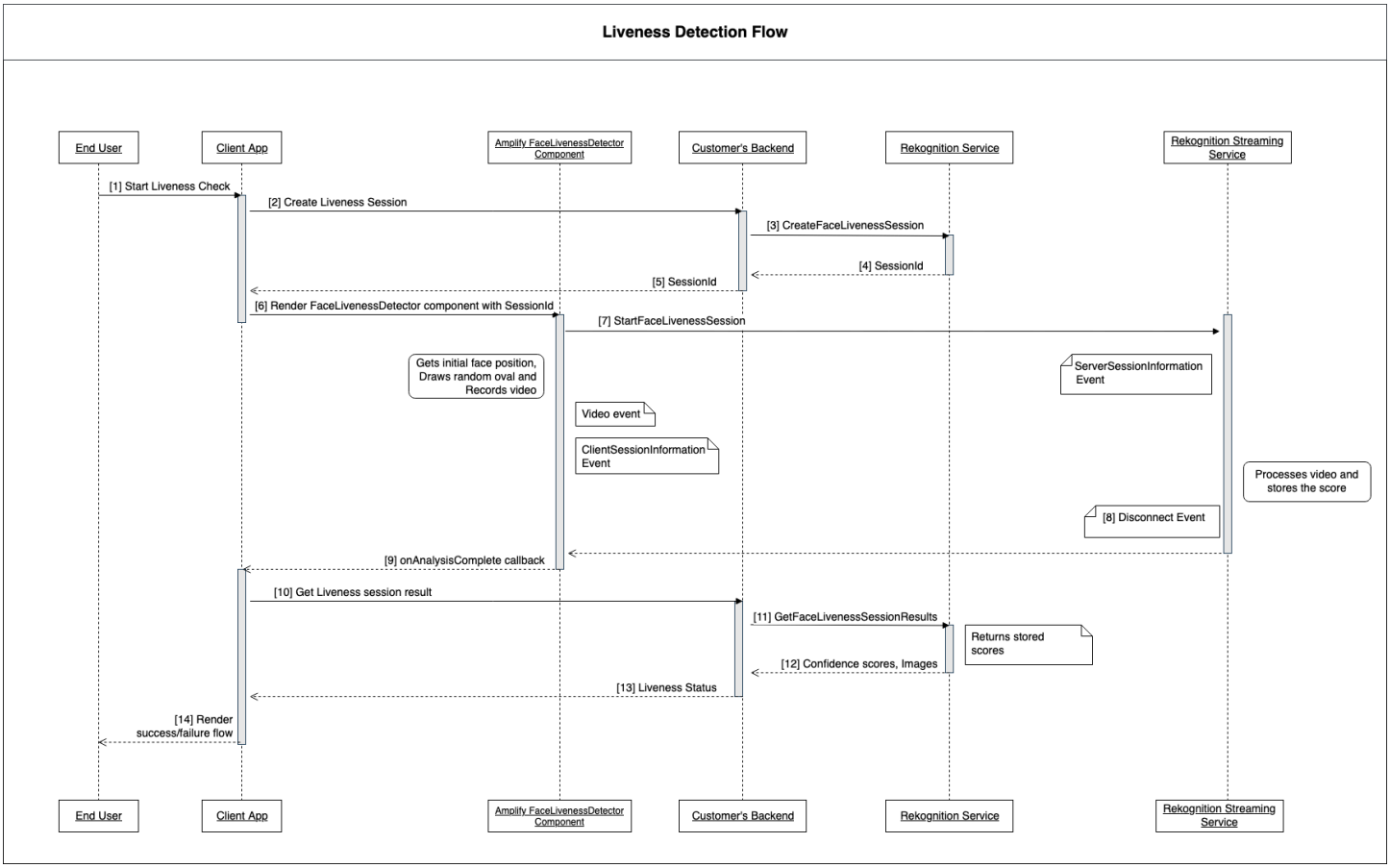
- 彩色摄像头：前置摄像头应该能够记录色彩。
- 没有虚拟摄像头或摄像头软件。
- 最低录制能力：每秒 15 帧。
- 最低视频录制分辨率：480x640px。
- 当用户使用带有桌面的网络摄像头进行 Face Liveness 检查时，请务必将摄像头安装在开始 Face Liveness 检查的同一屏幕的顶部。

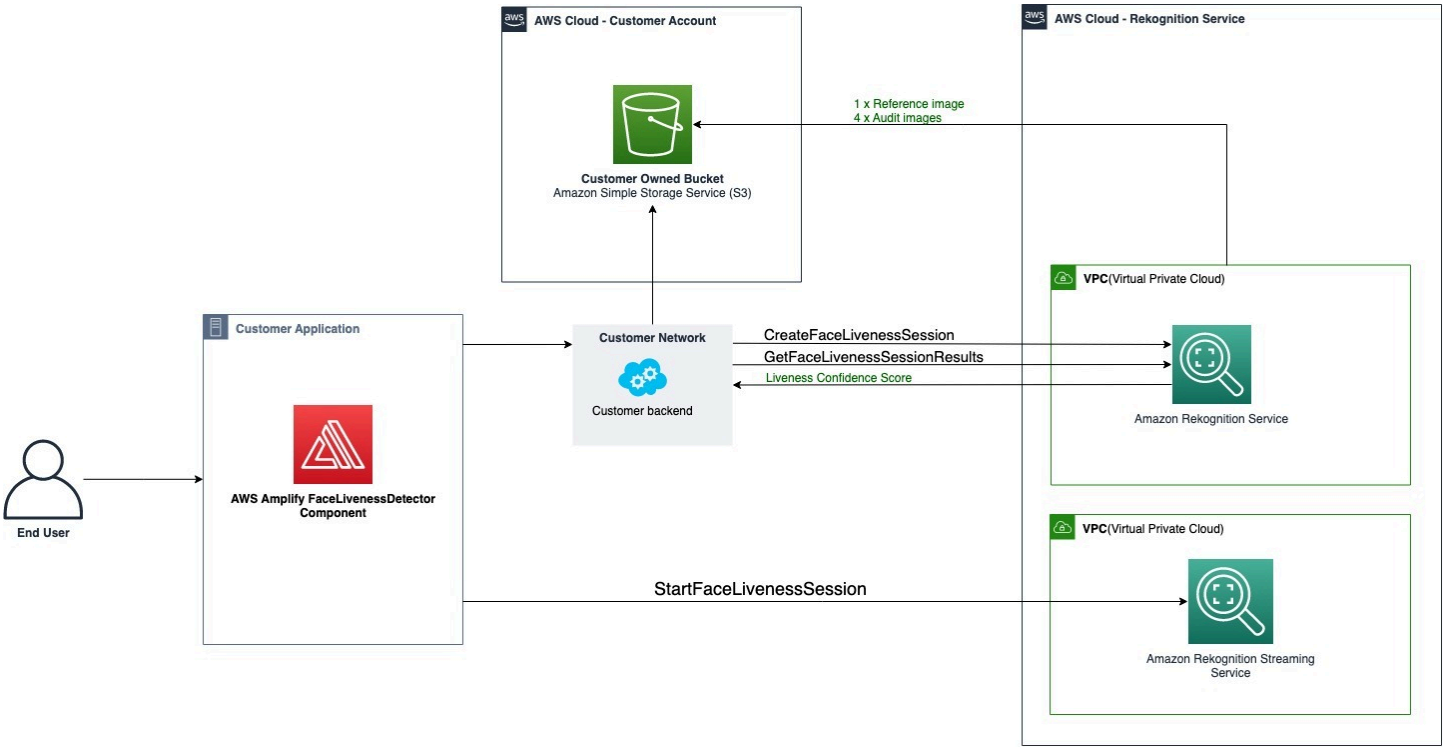
最低带宽要求：100 kbps

支持的浏览器：主流浏览器的最新三个版本，例如 Google Chrome、Mozilla Firefox、Apple Safari 和 Microsoft Edge。有关浏览器支持的更多信息，请参阅支持将哪些浏览器与 AWS 管理控制台一起使用？

架构和序列图

下图详细介绍了 Amazon Rekognition Face Liveness 在该功能的架构和操作顺序方面的运作方式：





Face Liveness 检查过程包括以下几个步骤：

1. 用户在客户端应用程序中启动 Face Liveness 检查。
2. 客户端应用程序调用客户的后端，后端又调用 Amazon Rekognition 服务。该服务会创建一个 Face Liveness 会话并返回一个唯一的 SessionId。注意：发送后， SessionId 它会在 3 分钟后过期，因此只有 3 分钟的时间可以完成下面的步骤 3 到 7。每次 Face Liveness 检查都必须使用新的 sessionID。如果使用给定的 sessionID 进行后续的 Face Liveness 检查，则检查将失败。此外，a SessionId 将在发送 3 分钟后过期，这使得与会话关联的所有 Liveness 数据（例如 SessionID、参考图像、审核图像等）不可用。
3. 客户端应用程序使用 SessionId 获得的相应回调来渲染 Ampli FaceLivenessDetector fy 组件。
4. 该 FaceLivenessDetector 组件与 Amazon Rekognition 流媒体服务建立连接，在用户屏幕上呈现椭圆形，并显示一系列彩色灯光。 FaceLivenessDetector 实时录制视频并将其流式传输到亚马逊 Rekognition 流媒体服务。
5. Amazon Rekognition 流媒体服务实时处理视频，存储结果，并在流式传输完成后 DisconnectEvent 返回 FaceLivenessDetector 到组件。
6. 该 FaceLivenessDetector 组件调用`onAnalysisComplete`回调向客户端应用程序发出信号，表明直播已完成，分数已准备好检索。
7. 客户端应用程序调用客户的后端以获得一个布尔值标志，指示用户是否真实存在。客户后端向 Amazon Rekognition 服务请求获取置信度分数、参考和审核图片。客户后端使用这些属性来确定用户是否真实存在，并向客户端应用程序返回相应的响应。
8. 最后，客户端应用程序将响应传递给 FaceLivenessDetector组件，组件会相应地呈现 success/failure 消息以完成流程。

先决条件

使用 Amazon Rekognition Face Liveness 的先决条件包括以下内容：

1. 设置 AWS 账户
2. 设置 Face Liveness AWS SDKs
3. 设置 AWS Amplify 资源

步骤 1：设置 AWS 账户

如果您还没有 AWS 账户，请完成中所示的步骤创建 AWS 账户和用户来创建一个账户。

第 2 步：设置 Face Liveness AWS SDKs

如果您还没有，请安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

有几种方法可以对 AWS SDK 调用进行身份验证。本指南中的示例假设您在调用 AWS CLI 命令和 AWS SDK API 操作时使用默认凭据配置文件。

有关向您的用户账户授予对所选 AWS SDK 的访问权限的更多信息，请参阅授予编程访问权限页面。该页面还说明了如何在本地计算机上使用配置文件以及如何在 AWS 环境中运行示例代码。

确保调用 Face Liveness 操作的用户具有调用操作的正确权限，例如 `AmazonRekognitionFullAccess` 和 `AmazonS3FullAccess` 权限。

第 3 步：设置 AWS Amplify 资源

要将 Amazon Rekognition Face Liveness 集成到您的应用程序中，您必须将 Amplify SDK 设置 AWS 为使用 Amplify 组件。 FaceLivenessDetector

如果您还没有这样做，请按照 AWS CLI 入门中的说明设置 AWS 命令行界面 (AWS CLI)。安装 CLI 后，完成 Amplify 用户界面文档网站上显示的 “配置身份验证” 步骤以设置您的 Am AWS plify 资源。

检测 Face Liveness 的最佳实践

在使用 Amazon Rekognition Face Liveness 时，我们建议您遵循多种最佳实践。Face Liveness 最佳实践包括应在何处进行 Face Liveness 检查、使用审核图像以及选择置信度分数阈值的相关准则。

有关最佳实践的完整列表，请参阅Face Liveness 的使用建议。

对亚马逊 Rekognition Face Liveness 进行编程 APIs

要使用 Amazon Rekognition Face Liveness API，您必须创建一个执行以下步骤的后端：

1. 致电CreateFaceLivenessSession以启动 Face Liveness 会话。`CreateFaceLivenessSession` 操作完成后，用户界面会提示用户提交自拍视频。然后，AWS Amplify 的 FaceLivenessDetector 组件会调用StartFaceLivenessSession以执行活跃度检测。
2. 致电GetFaceLivenessSessionResults返回与 Face Liveness 会话相关的检测结果。
3. 按照 A mplify L iveness 指南中的步骤继续配置你的 React 应用程序以使用该 FaceLivenessDetector组件。

在使用 Face Liveness 之前，请确保您已经创建了 AWS 账户，设置了 AWS CLI 和 AWS SDKs，并设置了 AWS Amplify。您还应确保您的后端 API 的 IAM 策略具有涵盖以下内容的权限：`GetFaceLivenessSessionResults` 和 `CreateFaceLivenessSession`。有关更多信息，请参阅先决条件部分。

步骤 1: CreateFaceLivenessSession

CreateFaceLivenessSession API 操作会创建一个 Face Liveness 会话并返回一个唯一`SessionId`会话。

作为此操作输入的一部分，还可以指定 Amazon S3 存储桶的位置。这允许存储在 Face Liveness 会话期间生成的参考图像和审核图像。Amazon S3 存储桶必须位于调用者的 AWS 账户中，并且与 Face Livess 端点位于同一区域。此外，S3 对象密钥由 Face Liveness 系统生成。

也可以提供一个 `AuditImagesLimit`，这是一个介于 0 和 4 之间的数字。默认情况下，将它设置为 0。返回的图像数量是尽力而为，并以自拍视频的持续时间为准。

最后，你可以提供一个 ChallengePreference，包括你希望你的会话默认为的挑战类型和版本。默认情况下，这将使用 FaceLivenessMovementAndLightChallenge但可以设置为 FaceMovementChallenge。

请求示例

```
{
   "ClientRequestToken": "string",
   "KmsKeyId": "string",
   "Settings": {
      "AuditImagesLimit": number,
      "ChallengePreferences": [
         {
            "Type": "string",
            "Versions": {
               "Maximum": "string",
               "Minimum": "string"
            }
         }
      ],
      "OutputConfig": {
         "S3Bucket": "string",
         "S3KeyPrefix": "string"
      }
```

```
    }
}
```

响应示例

```
{
    {"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
}
```

步骤 2: StartFaceLivenessSession

CreateFaceLivenessSession API 操作完成后，AWS Amplify 组件将执行 StartFaceLivenessSession API 操作。系统会提示用户拍摄自拍视频。要成功进行检查，用户必须将脸部放在屏幕上的椭圆形内，同时保持良好的光线。有关更多信息，请参阅 Face Liveness 的使用建议。

此 API 操作需要在 Face Liveness 会话期间捕获的视频、从 API 操作中 CreateFaceLivenessSession 获得的 sessionID 以及`onAnalysisComplete`回调。回调可用于向后端发出调用 API 操作的信号，GetFaceLivenessSessionResults API 操作会返回置信度分数、参考和审计图像。

请注意，此步骤由客户端应用程序上的 AWS Amplify FaceLivenessDetector 组件执行。您不需要进行其他设置即可调用 `StartFaceLivenessSession`。

步骤 3: GetFaceLivenessSessionResults

GetFaceLivenessSessionResults API 操作检索特定 Face Liveness 会话的结果。它需要会话 ID 作为输入，并返回相应的 Face Liveness 置信度分数。它还提供包含面部边界框的参考图像，以及包含面部边界框的审核图像。Face Liveness 置信度分数介于 0—100 之间。

请求示例

```
{"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
```

响应示例

```
{
    "SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8",
    "Confidence": 98.9735,
```

```
    "ReferenceImage": {
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "file-name",
        },
        "BoundingBox": {
           "Height": 0.4943420886993408,
            "Left": 0.8435328006744385,
            "Top": 0.8435328006744385,
            "Width": 0.9521094560623169}
    },
    "AuditImages": [{
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "audit-image-name",
        },
        "BoundingBox": {
           "Width": 0.6399999856948853,
           "Height": 0.47999998927116394,
           "Left": 0.1644444465637207,
           "Top": 0.17666666209697723}
    }],
    "Status": "SUCCEEDED"
}
```

步骤 4：对结果做出回应

Face Liveness 会话结束后，将检查的置信度分数与指定的阈值进行比较。如果分数高于阈值，则用户可以转到下一个屏幕或任务。如果检查失败，将通知用户并提示其重试。

直言不讳地活着 APIs

你可以使用任何 AWS 支持的软件开发工具包测试 Amazon Rekognition Face Liveness，比如 AWS Python SDK Boto3 或适用于 Java 的 AWS 开发工具包。您可以使用所选的 SDK 调`GetFaceLivenessSessionResults` APIs 用`CreateFaceLivenessSession`和。下一节演示了如何使用 Python 和 Java 调 APIs 用它们 SDKs。

要称之为 Face Liveness： APIs

- 如果您还没有这样做，请创建或更新具有 `AmazonRekognitionFullAccess` 权限的用户。有关更多信息，请参阅步骤 1：设置 AWS 账户并创建用户。

- 如果您还没有，请安装和配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅步骤 2：设置 AWS CLI 和 AWS SDKs。

Python

以下代码段显示了如何在 Python 应用程序 APIs 中调用它们。请注意，要运行此示例，您需要使用不低于 1.26.110 版本的 Boto3 SDK，但建议使用最新版本的 SDK。

```
import boto3

session = boto3.Session(profile_name='default')
client = session.client('rekognition')

def create_session():

    response = client.create_face_liveness_session()

    session_id = response.get("SessionId")
    print('SessionId: ' + session_id)

    return session_id


def get_session_results(session_id):

    response = client.get_face_liveness_session_results(SessionId=session_id)

    confidence = response.get("Confidence")
    status = response.get("Status")

    print('Confidence: ' + "{:.2f}".format(confidence) + "%")
    print('Status: ' + status)

    return status


def main():
    session_id = create_session()
    print('Created a Face Liveness Session with ID: ' + session_id)

    status = get_session_results(session_id)
    print('Status of Face Liveness Session: ' + status)
```

```
if __name__ == "__main__":
    main()
```

Java

以下代码段显示了如何在 Java 应用程序 APIs 中调用它们：

```
package aws.example.rekognition.liveness;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionRequest;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

    public static void main(String[] args) throws Exception {

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);

            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);

        } catch(AmazonRekognitionException e) {
           e.printStackTrace();
        }
    }
```

```
    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }

    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);

        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Java V2

以下片段演示了如何 APIs 使用 AWS Java V2 SDK 调用 Face Liveness：

```
package aws.example.rekognition.liveness;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionRequest;
```

```
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

    public static void main(String[] args) throws Exception {

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);

            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);

        } catch(AmazonRekognitionException e) {
           e.printStackTrace();
        }
    }

    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }

    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
```

```
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);

        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Node.Js

以下片段演示了如何 AWS 使用 Node.Js SDK 调用 Face Liv APIs eness：

```
const Rekognition = require("aws-sdk/clients/rekognition");

const rekognitionClient = new Rekognition({ region: "us-east-1" });

async function createSession() {
    const response = await rekognitionClient.createFaceLivenessSession().promise();

    const sessionId = response.SessionId;
    console.log("SessionId:", sessionId);

    return sessionId;
}

async function getSessionResults(sessionId) {
    const response = await rekognitionClient
        .getFaceLivenessSessionResults({
            SessionId: sessionId,
        })
        .promise();

    const confidence = response.Confidence;
    const status = response.Status;
    console.log("Confidence:", confidence);
    console.log("Status:", status);
```

```
    return status;
}

async function main() {
    const sessionId = await createSession();
    console.log("Created a Face Liveness Session with ID:", sessionId);

    const status = await getSessionResults(sessionId);
    console.log("Status of Face Liveness Session:", status);
}

main();
```

Node.Js (Javascript SDK v3)

以下片段演示了如何 APIs 使用适用于 Javascript v3 AWS 的 Node.Js SDK 调用 Face Liveness：

```
import { RekognitionClient, CreateFaceLivenessSessionCommand } from "@aws-sdk/
client-rekognition"; // ES Modules
import const { RekognitionClient, CreateFaceLivenessSessionCommand } =
 require("@aws-sdk/client-rekognition"); // CommonJS import
const client = new RekognitionClient(config);
const input = {
  KmsKeyId: "STRING_VALUE",
  Settings: {
    OutputConfig: { // LivenessOutputConfig
      S3Bucket: "STRING_VALUE", // required
      S3KeyPrefix: "STRING_VALUE",
    },
    AuditImagesLimit: Number("int"),
  },
  ClientRequestToken: "STRING_VALUE",
};
const command = new CreateFaceLivenessSessionCommand(input);
const response = await client.send(command);
// { // CreateFaceLivenessSessionResponse
//   SessionId: "STRING_VALUE", // required
// };
```

配置和自定义您的应用程序

配置应用程序

您的 Face Liveness 应用程序可以在移动设备或桌面网络浏览器上运行。您需要配置 Face Liveness 组件以与您选择的解决方案集成。您还必须确保您的应用程序有权使用设备的摄像头。Amplify Liveness 指南详细说明了如何：

- 安装和配置 AWS Amplify
- 导入并渲染组 FaceLivenessDetector 件
- 听取回调
- Render Amplify 示例错误消息

自定义您的应用程序

您可以使用 AWS Amplify 自定义活动应用程序的某些组件。

有关翻译的信息，请参阅 Amplify 身份验证器文档。

有关自定义 Amplify 组件和主题的信息，请参阅有关主题的 Amplify 文档。

Face Liveness 责任共担模式

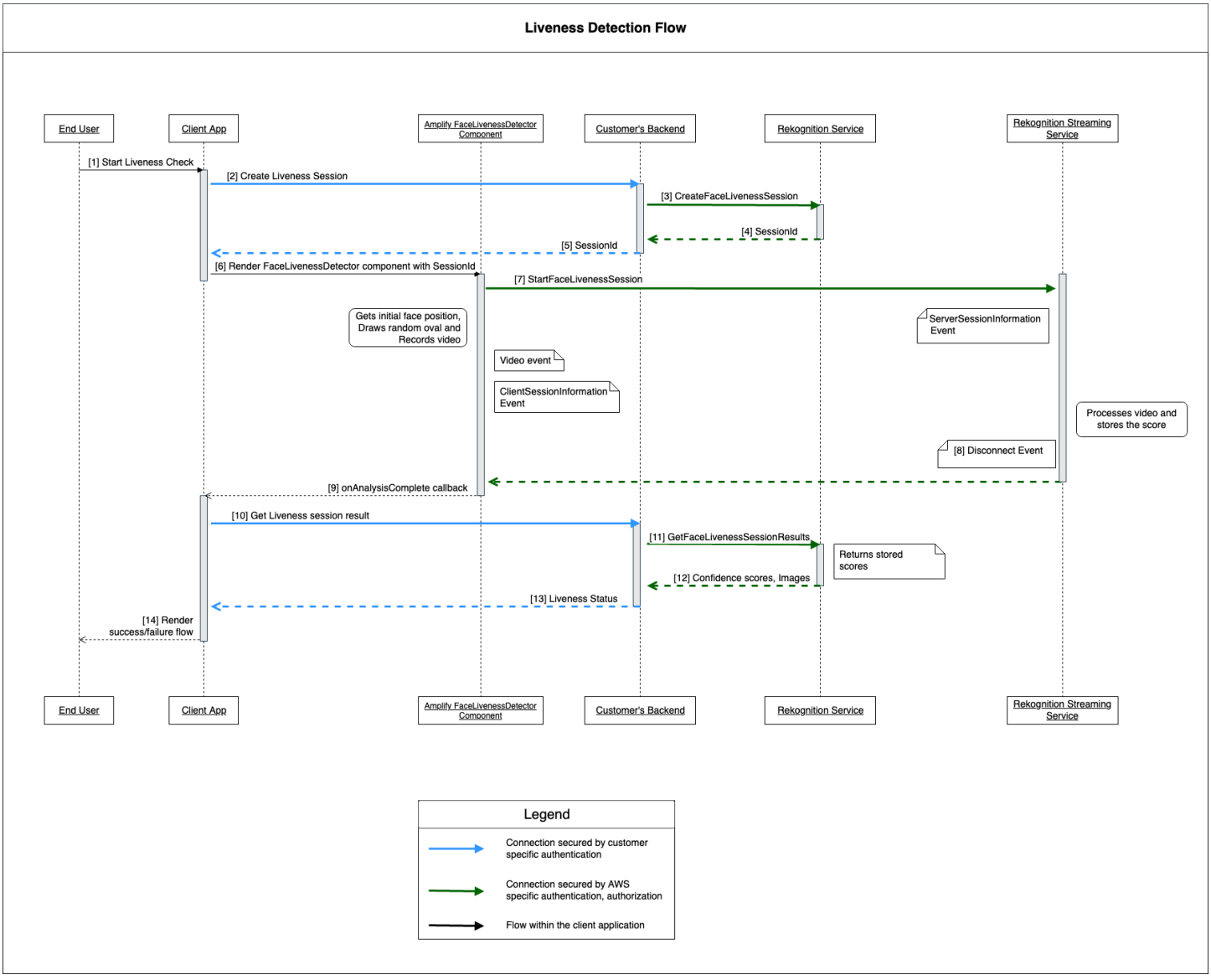
安全与合规是您（客户）的共同责任。 AWS 在此处阅读有关 AWS 分担责任模式的更多信息。

1. 对 AWS 服务的所有调用（通过客户端应用程序或客户后端）均通过身份验证（身份验证）进行 AWS AWS 身份验证和授权。Face Liveness 服务所有者负责确保这种情况发生。
2. 对客户后端的所有调用（来自客户端应用程序）均通过客户进行身份验证和授权。这一责任由客户承担。客户必须确保来自客户端应用程序的调用经过身份验证，并且不会以任何方式被篡改。
3. 客户后端必须识别执行 Face Liveness 挑战的终端用户。客户负责将终端用户与 Face Liveness 会话联系起来。Face Liveness 服务不区分终端用户。它只能识别来电身 AWS 份（由客户处理）。
4. 除了符合其用例要求和安全态势的 Face Liveness 之外，AWS 还建议客户进行额外的验证检查，例如位置地理定位（例如基于 IPOTPs）、一次性通行证代码 () 等。

“FaceMovementAndLightChallenge” 设置要求用户将脸向屏幕移动，并保持静止状态以获得一系列闪烁的灯光，从而为Rekognition Liveness提供最高的精度。我们建议客户使用此默认设置。或

者，客户可以启用 “FaceMovementChallenge” 设置，通过消除闪烁的灯光，将检查时间缩短几秒钟。虽然 “FaceMovementAndLightChallenge” 仍然是最大限度地提高准确性的最佳设置，但 “FaceMovementChallenge” 允许客户优先考虑更快的活跃度检查。在这些设置之间进行选择时，客户应考虑其用例要求，包括预期的攻击类型、所需的错误接受率和错误拒绝率，还应实施其他检查，例如地理位置（例如基于IP）、一次性通行码（OTPs）等。客户应在使用不同的置信度分数阈值测试 Liveness 性能后做出此决定，具体取决于他们的用例。客户有责任实施控制措施来保护发送视频的设备

以下流程图显示了哪些调用由 AWS 服务或客户进行身份验证：



对亚马逊 Rekognition Face Liveness 服务的所有呼叫均 AWS 受身份验证（使用签名机制）保护。AWS 这包括以下这些调用：

- [3] CreateFaceLivenessSessionAPI 调用（来自客户的后端）
- [7] StartFaceLivenessSessionAPI 调用（来自客户端应用程序）
- [11] GetFaceLivenessSessionResultsAPI 调用（来自客户的后端）

对客户后端的所有调用都需要有身份验证和授权机制。客户需要确保所code/library/etc使用的第三方得到积极维护和开发。客户还需要确保正确的终端用户调用正确的 Face Liveness 会话。客户必须对以下流程进行身份验证和授权：

- [2] 创建 Face Liveness 会话（来自客户端应用程序）
- [10] 获取 Face Liveness 会话结果（来自客户端应用程序）

客户可以遵循 STRIDE 安全模型，以确保其 API 调用受到保护。

类型	描述	安全控制
欺骗	威胁操作旨在访问和使用其他用户的证书，例如用户名和密码。	身份验证
篡改	意图恶意更改或修改永久数据的威胁行动。示例包括数据库中的记录，以及通过开放网络（例如互联网）在两台计算机之间传输的数据的更改。	完整性
否认	威胁操作旨在在缺乏跟踪操作能力的系统中执行违禁操作。	不可否认性
信息披露	威胁操作旨在读取未被授权访问的文件或读取传输中的数据。	机密性
拒绝服务	试图拒绝有效用户访问的威胁操作，例如使 Web 服务器暂时不可用或无法使用。	可用性

权限提升	威胁行动，旨在获得对资源的访问权限，以获得未经授权的信息访问权限或危害系统。	授权

AWS 通过以下方式保护其连接：

1. 计算请求签名，然后在服务端验证签名。使用此签名对请求进行身份验证。
2. AWS 客户需要设置适当的 IAM 角色才能授权某些操作/操作。调用 AWS 服务需要这些 IAM 角色。
3. 只允许 HTTPS AWS 服务请求。在开放网络中使用 TLS 对请求进行加密。这样可以保护请求的机密性 并保持请求的完整性。
4. AWS 服务记录足够的数据以识别客户拨打的电话。这样可以防止否认 攻击。
5. AWS 服务拥有保持足够的可用性

客户负责通过以下方式保护其服务和 API 调用：

1. 客户必须确保他们遵循正确的身份验证机制。有多种身份验证机制可用于对请求进行身份验证。客户可以探索基于摘要的身份验证 OAuth 、OpenID 连接和其他机制。
2. 客户必须确保其服务支持用于调用服务 API 的正确加密通道（如 TLS/HTTPS）。
3. 客户必须确保记录必要的数据，以便唯一识别 API 调用和调用者。他们应该能够使用定义的参数和调用时间识别调用其 API 的客户端。
4. 客户必须确保其系统可用，并保护他们免受 DDoS 攻击。以下是一些防御 DDo S 攻击的防御技巧的示例。

客户有责任保留其应用程序 up-to-date。有关更多信息，请参阅 Face Liveness 更新准则。

Face Liveness 更新准则

AWS 定期更新 Face Liveness AWS SDKs（用于客户后端）和 SDKs Amp AWS lify 的 FaceLivenessDetector 组件（用于客户端应用程序），以提供新功能、 APIs更新、增强的安全性、错误修复、可用性改进等。我们建议您保留 SDKs up-to-date 以确保该功能发挥最佳功能。如果您继续使用旧版本的 SDKs，则出于可维护性和安全原因，请求可能会被阻止。

Face Liveness 要求你使用 SDKs Am AWS plify 中包含的 FaceLivenessDetector 组件（React、iOS、Android）。

版本控制和时间范围

我们正在对 Face Liveness 功能的以下关键组件进行版本控制。我们遵循语义版本控制格式。例如，X.Y.Z 的版本格式，其中，X 代表主要版本，Y 代表次要版本，Z 代表补丁版本。

- Face Liveness 用户挑战（例如FaceMovementAndLightChallenge挑战）是 API 的一部分 StartFaceLivenessSession
- FaceLivenessDetector 通过 AWS Amplify 交付的组件 SDKs 用于客户端应用程序

主要版本：主要版本更新内容为关键安全性、重要 API 和引人注目的可用性更新。要继续使用 Face Liveness 功能，必须尽快更新应用程序和客户后端。一旦我们发布了新的主要版本，我们将在自新版本发布之日起 120 天内支持之前的主要版本。我们可能会在 120 天后屏蔽来自先前主要版本的请求。

次要版本：次要版本更新内容为重要的安全性和可用性功能及改进。我们强烈建议您应用这些更新。虽然我们努力确保次要更新尽可能长时间地向后兼容，但我们可能会在新的次要版本发布 180 天后宣布end-of-support之前的次要版本。

补丁版本：补丁版本更新内容为可选的错误修复和改进。虽然我们建议您保留您的版本 up-to-date以获得最佳的安全性和用户体验，但在我们发布新的主要或次要版本之前，我们会努力确保补丁更新完全向后兼容。

版本控制时段（主要版本为 120 天，次要版本为 180 天）适用于更新应用程序中的 SDK、将您的应用程序上传到应用商店或网站以及下载最新版本应用程序的用户。

版本发布和兼容性矩阵

针对 FaceLivenessDetector 组件或用户挑战赛的主要版本的发布通常是同时发布的。为帮助您跟踪版本依赖关系，请参阅下表中链接的资源。

SDK 版本和变更日志：

FaceLivenessDetector 适用于网页 SDK		FaceLivenessDetector 适用于 iOS 开发工具包	FaceLivenessDetector 适用于安卓 SDK
当前版本	变更日志	当前版本/变更日志	当前版本/变更日志

用户挑战：

挑战名称	版本	发行日期	退休日期
FaceMovementAndLightChallenge	v1.0.0	2023 年 4 月 10 日	不适用
FaceMovementChallenge	v1.0.0	2025 年 4 月 30 日	不适用

新版本的沟通

AWS 通过以下渠道传达新版本：

- 向与 Face Liveness 账户 ID 关联的账户电子邮件发送的服务运行状况更新电子邮件通知。
- 在相应 GitHub 存储库中发布的更新 AWS SDKs 和相关通知。
- 在相应 GitHub 存储库中发布了 AWS Amplify 的更新 SDKs 和相关通知。

我们建议您订阅这些频道以保持不变 up-to-date。

Face Liveness 常见问题

使用以下常见问题解答来查找有关 Rekognition Face Liveness 的常见问题的答案。

- 面部活跃度检查的输出是什么？

  Rekognition Face Liveness 为每次活跃度检查提供以下输出：

  - 置信度分数：返回介于 0 到 100 之间的数字分数。这个分数表明自拍视频很可能来自真实人物，而不是使用仿冒的不法分子。
  - 高质量图像：从自拍视频中提取了一张高质量的图像。该框架可用于各种用途，例如人脸比较、年龄估算或人脸搜索。
  - 审核图像：自拍视频最多返回四张图像，可用于审计跟踪记录。
- Rekognition Face Liveness 是否符合 iBeta 演示攻击检测 (PAD) 测试？

  iBeta Quality Assurance 的演示攻击检测 (PAD) 测试是根据 ISO/IEC 30107-3 进行的。iBeta 已获得 NIST/NVLAP 的认证，可以测试并提供符合此 PAD 标准的结果。Rekognition Face Liveness 以完美

的 PAD 分数通过了 1 级和 2 级 iBeta 演示攻击检测 (PAD) 一致性测试。可以在此处的 iBeta 网页上找到报告。

- 我怎样才能获得高质量的帧和额外帧？

  根据您的 CreateFaceLivenessSessionAPI 请求的配置，高质量帧和其他帧可以作为原始字节返回或上传到您指定的 Amazon S3 存储桶。

- 我可以更改椭圆形灯和彩色灯光的位置吗？

  不是。 椭圆形位置和彩色灯光是可实现更高精度的功能，因此无法自定义。

- 我可以根据我们的应用程序自定义用户界面吗？

  是的，您可以自定义大多数屏幕组件，例如主题、颜色、语言、文本内容和字体，使其与您的应用程序保持一致。有关如何自定义这些组件的详细信息，可以在我们的 React、Swift 和 Android 用户界面组件的文档中找到。

- 我可以自定义倒计时和时间，以便使人脸恰好置于椭圆形内吗？

  不，倒计时时间和脸部贴合时间是根据对1000多名用户的大规模内部研究预先确定的，目标是在两者之间取得最佳平衡，从而实现更高的准确性和延迟。因此，无法自定义这些时间设置。

- 不同颜色的灯光是否符合无障碍指南？

  是的，我们产品中的不同颜色的灯光符合 WCAG 2.1 中概述的无障碍指南。经过 1000 多次用户检查，用户体验每秒显示大约两种颜色，这符合将颜色限制在每秒三种的建议。这降低了大多数人群发作癫痫的可能性。

- SDK 是否会调整屏幕亮度以获得最佳效果？

  启动检查后，Face Liveness 手机 SDKs（适用于 Android 和 iOS）会自动调整亮度。但是，对于 Web SDK，网页存在一些限制，无法自动调整亮度。在这种情况下，我们希望 Web 应用程序指示终端用户手动提高屏幕亮度以获得最佳效果。

- 它必须是椭圆形吗？ 我们可以用其他类似的形状吗？

  不可以，椭圆形的大小、形状和位置不可自定义。精心挑选了特殊的椭圆形设计，因为它在准确捕捉和分析面部动作方面非常有效。因此，无法修改椭圆形状。

- end-to-end延迟是多少？

我们测量从用户开始执行完成活跃度检查所需的操作到用户获得结果（通过或失败）的 end-to-end 延迟。在最佳情况下，延迟为 5 秒。在平均情况下，我们预计约为 7 秒。在最坏情况下，延迟为 11 秒。我们看到 end-to-end延迟会有所不同，因为它取决于：用户完成所需操作（即将脸部移入椭圆形）的时间、网络连接、应用程序延迟等。

- 我可以在没有 Amplify SDK 的情况下使用 Face Liveness 功能吗？

  不可以，使用 Rekognition Face Liveness 功能需要 Amplify SDK。

- 在哪里可以找到与 Face Liveness 相关的错误状态？

  您可以在这里看到不同的 Face Liveness 错误状态。

- Face Liveness 在我所在的地区不可用。如何使用该功能？

  您可以选择在任何可用的地区调用 Face Liveness，具体取决于您的流量负荷和距离。Face liveness 目前在以下 AWS 地区可用：

  - 美国东部（弗吉尼亚州北部）
  - 美国西部（俄勒冈州）
  - 欧洲地区（爱尔兰）
  - 亚太地区（东京、孟买）

  即使您的 AWS 账户位于不同的区域，延迟差异预计也不会很大。您可以通过 Amazon S3 位置或原始字节获取高质量的自拍框和审核图像，但您的 Amazon S3 存储桶必须与 Face Liveness 的 AWS 区域匹配。如果它们不同，则必须以原始字节的形式接收图像。

- Amazon Rekognition Liveness Detection 是否使用客户内容来改进服务？

  通过使用 AWS Organizations 的选择退出政策，您可以选择不使用您的图像和视频输入来提高或开发 Rekognition 和其他 Amazon 机器学习/人工智能技术的质量。有关如何选择退出的信息，请参阅管理 AI 服务选择退出政策。

批量分析

Amazon Rekognition 批量分析允许您在操作中使用清单文件异步处理大量图像。StartMediaAnalysisJob每张图像的输出都与您用于分析的操作返回的输出相匹配。

目前，Rekognition 支持通过该操作进行分析。DetectModerationLabels

您需要为任务成功处理的图像数量付费。已完成任务的结果将输出到指定的 Amazon S3 存储桶。

请注意，批量分析不支持 Amazon A2I 集成。

API 可以检测动画或插图内容类型，有关检测到的内容类型的信息将作为响应的一部分返回。

主题

- 批量处理图像
- 批量分析输出清单
- 内容类型
- 验证预测和训练适配器

批量处理图像

您可以通过提交清单文件并调用 StartMediaAnalysisJob 操作来启动新的批量分析作业。输入清单文件包含对 Amazon S3 存储桶中图像的引用，其格式如下：

```
{"source-ref": "s3://foo/bar/1.jpg"}
```

创建批量分析任务 (CLI)

1. 如果您尚未执行以下操作，请：
   a. 使用 AmazonRekognitionFullAccess 和 AmazonS3ReadOnlyAccess 权限创建或更新用户。有关更多信息，请参阅 步骤 1：设置 AWS 账户并创建用户。
   b. 安装并配置 AWS CLI 和 AWS SDKs。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。
2. 将图像上传到 S3 存储桶。

有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

3. 使用以下命令创建和检索批量分析任务。

CLI

使用以下命令调用该StartMediaAnalysisJob操作进行分析： DetectModerationLabels

```
# Requests
# Starting DetectModerationLabels job with default settings
aws rekognition start-media-analysis-job \
--operations-config "DetectModerationLabels={MinConfidence='1'}" \
--input "S3Object={Bucket=amzn-s3-demo-source-bucket,Name=my-input.jsonl}" \
--output-config "S3Bucket=amzn-s3-demo-destination-bucket;,S3KeyPrefix=my-results"
```

您可以使用GetMediaAnalysisJob操作获取有关给定任务的信息，例如存储结果和摘要文件的存储桶的 Amazon S3 路径。您向其提供 StartMediaAnalysisJob 或返回的任务 ID ListMediaAnalysisJob。有关各个任务的详细信息仅保留一年。

```
# Request
aws rekognition get-media-analysis-job \
--job-id customer-job-id
```

您可以使用任务操作列出所有批量分析，该操作会返回ListMediaAnalysisJobs任务页面。使用 `max-results` 参数，您可以指定每页返回的最大作业数，值不超过 `max-results`。每页最多返回 100 个结果。有关各个任务的详细信息仅保留一年。

```
# Request
# Specify number of jobs to return per page, limited to max-results.
aws rekognition list-media-analysis-jobs --max-results 1
```

批量分析输出清单

批量分析任务会生成一个包含任务结果的输出清单文件，以及一个清单摘要，其中包含处理输入清单条目时出现的任何错误的统计数据和详细信息。

如果输入清单中包含重复的条目，则该任务不会尝试筛选出唯一的输入，而是会处理所有提供的条目。

输出清单文件的格式如下：

```
// Output manifest for content moderation
{"source-ref":"s3://foo/bar/1.jpg", "detect-moderation-labels":
 {"ModerationLabels":[],"ModerationModelVersion":"7.0","ContentTypes":
[{"Confidence":72.7257,"Name":"Animated"}]}}
```

输出清单摘要的格式如下：

```
{
    "version": "1.0",               # Schema version, 1.0 for GA.
    "statistics": {
        "total-json-lines": Number,  # Total number json lines (images) in the input
 manifest.
        "valid-json-lines": Number,  # Total number of JSON Lines (images) that contain
 references to valid images.
        "invalid-json-lines": Number # Total number of invalid JSON Lines. These lines
 were not handled.
    },
    "errors": [
        {
            "line-numer": Number,   # The number of the line in the manifest where the
 error occured.
            "source-ref": "String", # Optional. Name of the file if was parsed.
            "code": "String",       # Error code.
            "message": "String"     # Description of the error.
        }
     ]
}
```

内容类型

操作会返回有关 StartMediaAnalysisJob 操作所分析的媒体内容类型的信息。 GetMediaAnalysisJob ContentType 可以是两个不同的类别之一：

- 动画内容，包括视频游戏和动画（例如卡通、连环画、漫画、动画）。
- 插图内容，包括素描、绘画和速写。

验证预测和训练适配器

还可以通过 Rekognition 控制台利用批量分析来获取一批图像的预测结果，验证这些预测，然后使用经过验证的预测创建适配器。适配器允许您提高任何受支持的 Rekognition 操作的准确性。

目前，您可以创建适配器以与 Rekognition 自定义审核特征配合使用。通过创建适配器并将其提供给DetectModerationLabels操作，您可以提高与您的特定用例相关的内容审核任务的准确性。

有关自定义审核的更多信息，请参阅使用自定义审核提高准确性。有关如何验证使用批量分析所做的预测的说明，请参阅批量分析和验证。有关如何使用 Rekognition 控制台验证预测和创建适配器的教程，请参阅自定义审核适配器教程。

教程

这些跨服务教程演示了如何使用 Rekognition 的 API 操作以及其他 AWS 服务来创建示例应用程序和完成各种任务。这些教程大多使用 Amazon S3 来存储图像或视频。其他常用的服务包括 AWS Lambda。

主题

- 使用 Amazon RDS 和 DynamoDB 存储 Amazon Rekognition 数据
- 使用 Amazon Rekognition 和 Lambda 标记 Amazon S3 存储桶中的资产
- 创建 AWS 视频分析器应用程序
- 创建 Amazon Rekognition Lambda 函数
- 使用 Amazon Rekognition 进行身份验证
- 使用 Amazon Rekognition（REK）检测用于营销应用程序的标签
- 使用 Lambda 和 Python 检测图像中的标签

使用 Amazon RDS 和 DynamoDB 存储 Amazon Rekognition 数据

在使用 Amazon Rekognition 时 APIs，请务必记住，API 操作不会保存任何生成的标签。您可以通过将这些标签以及相应图像的标识符放入数据库来保存这些标签。

本教程演示如何检测标签并将检测到的标签保存到数据库中。本教程中开发的示例应用程序将从 Amazon S3 存储桶读取图像，对这些图像调用DetectLabels操作，并将生成的标签存储在数据库中。该应用程序将数据存储在 Amazon RDS 数据库实例或 DynamoDB 数据库中，具体取决于您要使用的数据库类型。

您将使用 AWS SDK for Python 或本教程。您还可以查看 AWS 文档 SDK 示例存储库，了解更多 GitHub P ython 教程。

主题

- 先决条件
- 获取 Amazon S3 存储桶中的图像的标签
- 创建 Amazon DynamoDB 表
- 将数据上传到 DynamoDB
- 在 Amazon RDS 中创建 MySQL 数据库

- 将数据上传到 Amazon RDS MySQL 表

先决条件

在开始本教程之前，你需要安装 Python 并完成设置 Python AWS 开发工具包所需的步骤。除此之外，请确保您：

已创建一个 AWS 账户和一个 IAM 角色

已安装 Python SDK (Boto3)

正确配置您的 AWS 访问凭证

已创建 Amazon S3 存储桶，里面填充了图像

如果使用 RDS 存储数据，则已创建 RDS 数据库实例

获取 Amazon S3 存储桶中的图像的标签

首先编写一个函数，该函数将获取您的 Amazon S3 存储桶中图像的名称并检索该图像。将显示此图像以确认是否将正确的图像传递到函数中也DetectLabels包含的调用中。

1. 找到您要使用的 Amazon S3 存储桶，并记下其名称。您将调用此 Amazon S3 存储桶并读取其中的图像。确保您的存储桶包含一些要传递给DetectLabels操作的图像。
2. 编写代码以连接到 Amazon S3 存储桶。您可以使用 Boto3 连接 Amazon S3 资源，从 Amazon S3 存储桶中检索图像。连接到 Amazon S3 资源后，您可以通过为 Amazon S3 存储桶名称提供存储桶方法来访问您的存储桶。连接到 Amazon S3 存储桶后，您可以使用对象方法从存储桶中检索图像。通过使用 Matplotlib，您可以使用此连接在图像处理时对其进行可视化。Boto3 还用于连接到 Rekognition 客户端。

   在以下代码中，在 region_name 参数中提供您的区域。您需要将 Amazon S3 存储桶名称和图像名称传递给 DetectLabels，这将返回相应图像的标签。仅从响应中选择标签后，将返回图像名称和标签。

```
import boto3
from io import BytesIO
from matplotlib import pyplot as plt
from matplotlib import image as mp_img
```

```
boto3 = boto3.Session()

def read_image_from_s3(bucket_name, image_name):

    # Connect to the S3 resource with Boto 3
    # get bucket and find object matching image name
    s3 = boto3.resource('s3')
    bucket = s3.Bucket(name=bucket_name)
    Object = bucket.Object(image_name)

    # Downloading the image for display purposes, not necessary for detection of
 labels
    # You can comment this code out if you don't want to visualize the images
    file_name = Object.key
    file_stream = BytesIO()
    Object.download_fileobj(file_stream)
    img = mp_img.imread(file_stream, format="jpeg")
    plt.imshow(img)
    plt.show()

    # get the labels for the image by calling DetectLabels from Rekognition
    client = boto3.client('rekognition', region_name="region-name")
    response = client.detect_labels(Image={'S3Object': {'Bucket': bucket_name,
 'Name': image_name}},
                                    MaxLabels=10)

    print('Detected labels for ' + image_name)

    full_labels = response['Labels']

    return file_name, full_labels
```

3. 将此代码保存在名为 get_images.py 的文件中。

创建 Amazon DynamoDB 表

以下代码使用 Boto3 连接到 DynamoDB，并使用 DynamoDB `CreateTable` 方法创建名为 Images 的表。该表具有一个由分区键 (Image) 和排序键 (Labels) 组成的复合主键。Image 键包含图像的名称，而 Labels 键存储分配给该图像的标签。

```
import boto3
```

```
def create_new_table(dynamodb=None):
    dynamodb = boto3.resource(
        'dynamodb',)
    # Table defination
    table = dynamodb.create_table(
        TableName='Images',
        KeySchema=[
            {
                'AttributeName': 'Image',
                'KeyType': 'HASH'  # Partition key
            },
            {
                'AttributeName': 'Labels',
                'KeyType': 'RANGE'  # Sort key
            }
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'Image',
                'AttributeType': 'S'
            },
            {
                'AttributeName': 'Labels',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )
    return table

if __name__ == '__main__':
    device_table = create_new_table()
    print("Status:", device_table.table_status)
```

将此代码保存到编辑器中，运行一次即可创建 DynamoDB 表。

将数据上传到 DynamoDB

现在 DynamoDB 数据库已经创建完毕，并且您有了获取图像标签的函数，您可以将标签存储在 DynamoDB 中。以下代码检索 S3 存储桶中的所有图像，为它们获取标签，然后将数据存储在 DynamoDB 中。

1. 您需要编写用于将数据上传到 DynamoDB 的代码。名为`get_image_names`的函数用于连接您的 Amazon S3 存储桶，它以列表形式返回存储桶中所有图像的名称。您需要将此列表传递到`read_image_from_S3` 函数中，该函数是从您创建的`get_images.py` 文件中导入的。

```
import boto3
import json
from get_images import read_image_from_s3

boto3 = boto3.Session()

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. 我们之前创建的`read_image_from_S3` 函数将返回正在处理的图像的名称以及与该图像关联的标签字典。名为`find_values`的函数用于从响应中仅获取标签。然后，可以将图像的名称及其标签上传到您的 DynamoDB 表中。

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. 您将使用名为load_data的第三个函数将图像和标签实际加载到您创建的 DynamoDB 表中。

```
def load_data(image_labels, dynamodb=None):

    if not dynamodb:
        dynamodb = boto3.resource('dynamodb')

    table = dynamodb.Table('Images')

    print("Adding image details:", image_labels)
    table.put_item(Item=image_labels)
    print("Success!!")
```

4. 这里是调用我们之前定义的三个函数以及执行操作的地方。将上面定义的三个函数以及下面的代码添加到 Python 文件中。运行该代码。

```
bucket = "bucket_name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json_string = json.dumps(image_labels, indent=4)
    labels=set(find_values("Name", image_json_string))
    print("Labels found: " + str(labels))
    labels_dict = {}
    print("Saving label data to database")
    labels_dict["Image"] = str(image_name)
    labels_dict["Labels"] = str(labels)
    print(labels_dict)
    load_data(labels_dict)
    print("Success!")
```

您刚刚为图像生成标签并将这些标签存储在 DynamoDB 实例中。DetectLabels在学习本教程时，请务必删除您创建的所有资源。这样可以防止您因未使用的资源而被收取费用。

在 Amazon RDS 中创建 MySQL 数据库

在继续操作之前，请确保您已经完成了 Amazon RDS 的设置过程并使用 Amazon RDS 创建了一个 MySQL 数据库实例。

以下代码使用了 PyMySQL 库和您的 Amazon RDS 数据库实例。它会创建一个表来保存您的图像名称以及与这些图像关联的标签。Amazon RDS 会收到用于创建表和向表中插入数据的命令。要使用 Amazon RDS，您必须使用您的主机名、用户名和密码连接到 Amazon RDS 主机。通过向 PyMy SQL 的connect函数提供这些参数并创建游标实例，您将连接到 Amazon RDS。

1. 在以下代码中，将主机的值替换为您的 Amazon RDS 主机端点，并将用户的值替换为与您的 Amazon RDS 实例关联的主用户名。您还需要将密码替换为主用户的主密码。

```
import pymysql

host = "host-endpoint"
user = "username"
password = "master-password"
```

2. 创建数据库和表格，将图像和标签数据插入其中。通过运行并提交创建查询来执行此操作。以下代码创建一个数据库。仅运行此代码一次。

```
conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

# run once
create_query = "create database rekogDB1"
print("Creation successful!")
cursor.execute(create_query)
cursor.connection.commit()
```

3. 数据库创建完成后，必须创建一个表来插入图像名称和标签。要创建表，首先要将 use SQL 命令以及数据库名称传递给execute函数。建立连接后，将运行创建表的查询。以下代码连接到数据库，然后创建一个表，其中包含一个名为image_id 的主键和一个存储标签的文本属性。使用您之前定义的导入和变量，然后运行此代码在数据库中创建表。

```
# connect to existing DB
cursor.execute("use rekogDB1")
cursor.execute("CREATE TABLE IF NOT EXISTS test_table(image_id VARCHAR (255)
 PRIMARY KEY, image_labels TEXT)")
conn.commit()
print("Table creation - Successful creation!")
```

将数据上传到 Amazon RDS MySQL 表

创建 Amazon RDS 数据库并在数据库中创建表后，您可以获取图像的标签并将这些标签存储在 Amazon RDS 数据库中。

1. 连接到您的 Amazon S3 存储桶并检索存储桶中所有图像的名称。这些图像名称将传递到您之前创建的 `read_image_from_s3` 函数中，以获取所有图像的标签。以下代码连接到您的 Amazon S3 存储桶，并返回存储桶中所有图像的列表。

```
import pymysql
from get_images import read_image_from_s3
import json
import boto3

host = "host-endpoint"
user = "username"
password = "master-password"

conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. 来自 DetectLabelsAPI 的响应不仅仅包含标签，因此请编写一个函数来仅提取标签值。以下函数返回一个仅包含标签的列表。

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
```

```
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. 您需要一个函数来将图像名称和标签插入表中。以下函数运行插入查询并插入任意一对给定的图像名称和标签。

```
def upload_data(image_id, image_labels):

    # insert into db
    cursor.execute("use rekogDB1")
    query = "INSERT IGNORE INTO test_table(image_id, image_labels) VALUES (%s, %s)"
    values = (image_id, image_labels)
    cursor.execute(query, values)
    conn.commit()
    print("Insert successful!")
```

4. 最后，您必须运行上面定义的函数。在以下代码中，将收集存储桶中所有图像的名称并提供给调用的函数DetectLabels。之后，标签及其适用的图像名称将上传到您的 Amazon RDS 数据库。将上面定义的三个函数以及下面的代码复制到 Python 文件中。运行 Python 文件。

```
bucket = "bucket-name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json = json.dumps(image_labels, indent=4)
    labels=set(find_values("Name", image_json))
    print("Labels found: " + str(labels))
    unique_labels=set(find_values("Name", image_json))
    print(unique_labels)
    image_name_string = str(image_name)
    labels_string = str(unique_labels)
    upload_data(image_name_string, labels_string)
    print("Success!")
```

您已成功使用 DetectLabels 为图像生成标签，并使用 Amazon RDS 将这些标签存储在 MySQL 数据库中。在学习本教程时，请务必删除您创建的所有资源。这样可以防止您因未使用的资源而被收取费用。

有关更多 AWS 多服务示例，请参阅 AWS 文档 SDK 示例GitHub 存储库。

使用 Amazon Rekognition 和 Lambda 标记 Amazon S3 存储桶中的资产

在本教程中，您将创建一个自动标记位于 Amazon S3 存储桶中的数字资产的 AWS Lambda 函数。Lambda 函数会读取给定 Amazon S3 存储桶中的所有对象。对于存储桶中的每个对象，它都会将图像传递给 Amazon Rekognition 服务以生成一系列标签。每个标签都用于创建应用于图像的标签。在您执行 Lambda 函数后，它会根据给定 Amazon S3 存储桶中的所有图像自动创建标签并将其应用于图像。

例如，假设您运行了 Lambda 函数，并且在 Amazon S3 存储桶中存储了这张图片。

然后，应用程序会自动创建标签并将其应用于图像。

Tags (6)

Track storage cost of other criteria by tagging your objects. **Learn more**

Key	Value
Nature	99.99188
Volcano	97.60948
Eruption	96.54574
Lava	79.63064
Mountain	99.99188
Outdoors	99.99188

> **Note**
>
> 您在本教程中使用的服务是 AWS 免费套餐的一部分。完成本教程后，我们建议终止您在教程中创建的所有资源，这样您无需为此付费。

本教程使用适用于 Java 的 AWS SDK 版本 2。有关其他 Java V2 教程，请参阅AWS 文档 SDK 示例 GitHub 存储库。

主题

- 先决条件
- 配置 IAM Lambda 角色
- 创建项目
- 编写代码
- 打包项目
- 部署 Lambda 函数
- 测试 Lambda 方法

先决条件

在开始之前，您需要完成设置 AWS SDK for Java 中的步骤。然后确保执行以下操作：

- Java 1.8 JDK。
- Maven 3.6 或更高版本。
- 一个含有 5-7 张自然图像的 Amazon S3 存储桶。这些图像由 Lambda 函数读取。

配置 IAM Lambda 角色

本教程使用 Amazon Rekognition 和 Amazon S3 服务。将 lambda 支持角色配置为具有允许其从 Lambda 函数调用这些服务的策略。

配置角色

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色，然后选择创建角色。

3. 选择 AWS 服务，然后选择 Lambda。
4. 选择 Permissions（权限）选项卡。
5. 搜索 AWSLambdaBasicExecutionRole。
6. 请选择下一个标签。
7. 选择审核。
8. 为角色命名 lambda 支持。
9. 选择创建角色。
10. 选择 lambda 支持以查看概述页面。
11. 选择附加策略。
12. AmazonRekognitionFullAccess从策略列表中选择。
13. 选择附加策略。
14. 搜索 A mazonS3 FullAccess，然后选择附加政策。

创建项目

创建一个新的 Java 项目，然后使用所需的设置和依赖项配置 Maven pom.xml。确保您的 pom.xml 文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</groupId>
<artifactId>WorkflowTagAssets</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>java-basic-function</name>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencyManagement>
    <dependencies>
        <dependency>
```

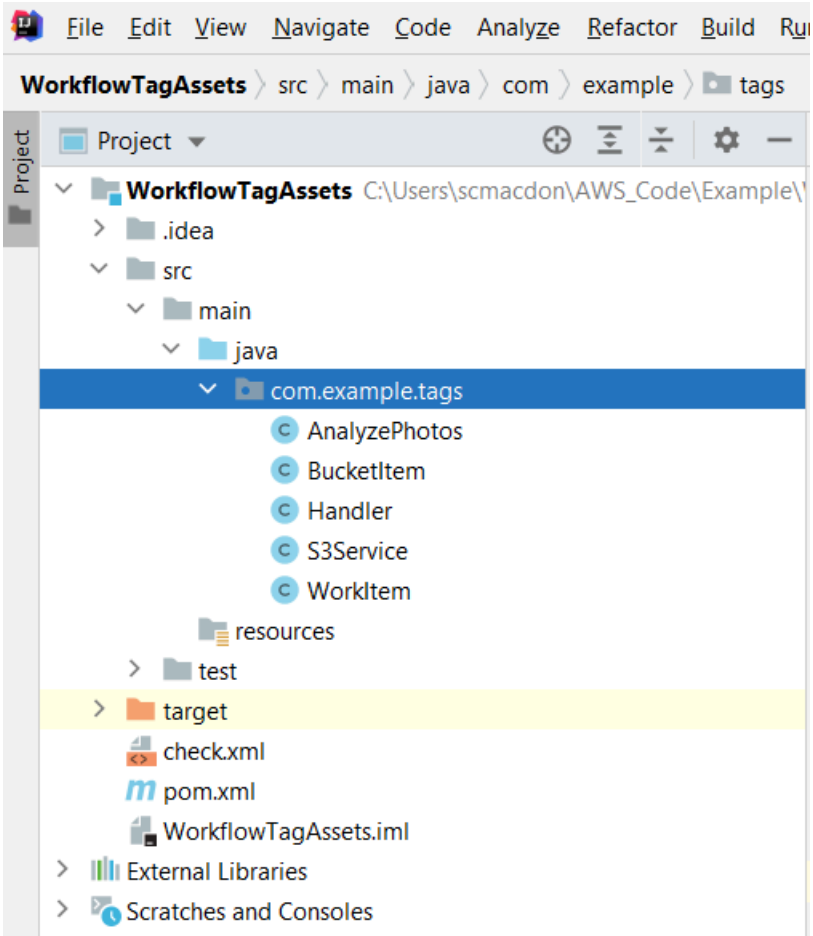
```
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>bom</artifactId>
            <version>2.10.54</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
   <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-lambda-java-core</artifactId>
        <version>1.2.1</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.10.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.13.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j18-impl</artifactId>
        <version>2.13.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
```

```
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>rekognition</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.2</version>
            <configuration>
                <createDependencyReducedPom>false</createDependencyReducedPom>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
```

```
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
  </build>
</project>
```

编写代码

使用 AWS Lambda 运行时 Java API 创建定义 Lambda 函数的 Java 类。在此示例中，Lambda 函数有一个名为 Handler 的 Java 类以及此用例所需的其他类。下图显示了项目中的 Java 类。请注意，所有 Java 类都位于名为 com.example.tags 的包中。



为代码创建以下 Java 类：

- Handl@@ er 使用 Lambda Java 运行时 API 并执行本教程中描述的用例。 AWS 执行的应用程序逻辑位于 handleRequest 方法中。

- S3Service 使用 Amazon S3 API 来执行 S3 操作。
- AnalyzePhotos使用 Amazon Rekognition API 来分析图片。
- BucketItem定义了存储 Amazon S3 存储桶信息的模型。
- WorkItem定义存储亚马逊 Rekognition 数据的模型。

处理程序类

此 Java 代码代表处理程序类。该类会读取传递到 Lambda 函数的标志。s3Service。ListBucketObjects方法返回一个 Lis t 对象，其中每个元素都是一个表示对象键的字符串值。如果标志值为真，则通过对列表进行迭代操作应用标签，并通过调用 s3Service.tagAssets 方法将标签应用于每个对象。如果标志值为假，则为 s 3Service。 deleteTagFrom调用删除标签的对象方法。另外，请注意，您可以使用LambdaLogger对象将消息记录到 Amazon CloudWatch 日志中。

> **Note**
>
> 确保将存储桶名称分配给 bucketName 变量。

```
package com.example.tags;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Handler implements RequestHandler<Map<String,String>, String> {

@Override
public String handleRequest(Map<String, String> event, Context context) {
    LambdaLogger logger = context.getLogger();
    String delFlag = event.get("flag");
    logger.log("FLAG IS: " + delFlag);
    S3Service s3Service = new S3Service();
    AnalyzePhotos photos = new AnalyzePhotos();

    String bucketName = "<Enter your bucket name>";
    List<String> myKeys = s3Service.listBucketObjects(bucketName);
    if (delFlag.compareTo("true") == 0) {
```

```
        // Create a List to store the data.
        List<ArrayList<WorkItem>> myList = new ArrayList<>();

        // loop through each element in the List and tag the assets.
        for (String key : myKeys) {

            byte[] keyData = s3Service.getObjectBytes(bucketName, key);

            // Analyze the photo and return a list where each element is a WorkItem.
            ArrayList<WorkItem> item = photos.detectLabels(keyData, key);
            myList.add(item);
        }

        s3Service.tagAssets(myList, bucketName);
        logger.log("All Assets in the bucket are tagged!");

    } else {

        // Delete all object tags.
        for (String key : myKeys) {
            s3Service.deleteTagFromObject(bucketName, key);
            logger.log("All Assets in the bucket are deleted!");
        }
     }
    return delFlag;
 }
}
```

S3Service 类

以下类使用 Amazon S3 API 执行 S3 操作。例如，该getObjectBytes方法返回一个表示图像的字节数组。同样，该listBucketObjects方法返回一个 Lis t 对象，其中每个元素都是指定键名的字符串值。

```
package com.example.tags;

import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
```

```
import software.amazon.awssdk.services.s3.model.ListObjectsResponse;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsRequest;
import java.util.ArrayList;
import java.util.List;
import software.amazon.awssdk.services.s3.model.Tagging;
import software.amazon.awssdk.services.s3.model.Tag;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectTaggingRequest;

public class S3Service {

private S3Client getClient() {

   Region region = Region.US_WEST_2;
   return S3Client.builder()
           .region(region)
           .build();
}

public byte[] getObjectBytes(String bucketName, String keyName) {

   S3Client s3 = getClient();

   try {

       GetObjectRequest objectRequest = GetObjectRequest
               .builder()
               .key(keyName)
               .bucket(bucketName)
               .build();

       // Return the byte[] from this object.
       ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
       return objectBytes.asByteArray();

   } catch (S3Exception e) {
       System.err.println(e.awsErrorDetails().errorMessage());
       System.exit(1);
   }
   return null;
}
```

```
// Returns the names of all images in the given bucket.
public List<String> listBucketObjects(String bucketName) {

   S3Client s3 = getClient();
   String keyName;

   List<String> keys = new ArrayList<>();

   try {
       ListObjectsRequest listObjects = ListObjectsRequest
               .builder()
               .bucket(bucketName)
               .build();

       ListObjectsResponse res = s3.listObjects(listObjects);
       List<S3Object> objects = res.contents();

       for (S3Object myValue: objects) {
           keyName = myValue.key();
           keys.add(keyName);
       }
       return keys;

   } catch (S3Exception e) {
       System.err.println(e.awsErrorDetails().errorMessage());
       System.exit(1);
   }
   return null;
}

// Tag assets with labels in the given list.
public void tagAssets(List myList, String bucketName) {

   try {

       S3Client s3 = getClient();
       int len = myList.size();

       String assetName = "";
       String labelName = "";
       String labelValue = "";

       // Tag all the assets in the list.
```

```
        for (Object o : myList) {

            // Need to get the WorkItem from each list.
            List innerList = (List) o;
            for (Object value : innerList) {

                WorkItem workItem = (WorkItem) value;
                assetName = workItem.getKey();
                labelName = workItem.getName();
                labelValue = workItem.getConfidence();
                tagExistingObject(s3, bucketName, assetName, labelName, labelValue);
            }
        }

   } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
   }
}

// This method tags an existing object.
private void tagExistingObject(S3Client s3, String bucketName, String key, String
label, String LabelValue) {

   try {

       // First need to get existing tag set; otherwise the existing tags are
overwritten.
       GetObjectTaggingRequest getObjectTaggingRequest =
GetObjectTaggingRequest.builder()
               .bucket(bucketName)
               .key(key)
               .build();

       GetObjectTaggingResponse response =
s3.getObjectTagging(getObjectTaggingRequest);

       // Get the existing immutable list - cannot modify this list.
       List<Tag> existingList = response.tagSet();
       ArrayList<Tag> newTagList = new ArrayList(new ArrayList<>(existingList));

       // Create a new tag.
       Tag myTag = Tag.builder()
               .key(label)
```

```
                .value(LabelValue)
                .build();

        // push new tag to list.
        newTagList.add(myTag);
        Tagging tagging = Tagging.builder()
                .tagSet(newTagList)
                .build();

        PutObjectTaggingRequest taggingRequest = PutObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .tagging(tagging)
                .build();

        s3.putObjectTagging(taggingRequest);
        System.out.println(key + " was tagged with " + label);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
 }

// Delete tags from the given object.
public void deleteTagFromObject(String bucketName, String key) {

    try {

        DeleteObjectTaggingRequest deleteObjectTaggingRequest =
DeleteObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .build();

        S3Client s3 = getClient();
        s3.deleteObjectTagging(deleteObjectTaggingRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
 }
```

```
}
```

AnalyzePhotos 班级

以下 Java 代码代表该AnalyzePhotos类。该类使用 Amazon Rekognition API 来分析图像。

```
package com.example.tags;

import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.ArrayList;
import java.util.List;

public class AnalyzePhotos {

 // Returns a list of WorkItem objects that contains labels.
 public ArrayList<WorkItem> detectLabels(byte[] bytes, String key) {

    Region region = Region.US_EAST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .region(region)
            .build();

    try {

        SdkBytes sourceBytes = SdkBytes.fromByteArray(bytes);

        // Create an Image object for the source image.
        Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

        DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
```

```
                .build();

        DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);

        // Write the results to a WorkItem instance.
        List<Label> labels = labelsResponse.labels();
        ArrayList<WorkItem> list = new ArrayList<>();
        WorkItem item ;
        for (Label label: labels) {
            item = new WorkItem();
            item.setKey(key); // identifies the photo.
            item.setConfidence(label.confidence().toString());
            item.setName(label.name());
            list.add(item);
        }
        return list;

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
    return null ;
  }
}
```

BucketItem 班级

以下 Java 代码表示存储 Amazon S3 对象数据的BucketItem类。

```
package com.example.tags;

public class BucketItem {

 private String key;
 private String owner;
 private String date ;
 private String size ;


 public void setSize(String size) {
    this.size = size ;
 }
```

```
 public String getSize() {
    return this.size ;
 }

 public void setDate(String date) {
    this.date = date ;
 }

 public String getDate() {
    return this.date ;
 }

 public void setOwner(String owner) {
    this.owner = owner ;
 }

 public String getOwner() {
    return this.owner ;
 }

 public void setKey(String key) {
    this.key = key ;
 }

 public String getKey() {
    return this.key ;
 }
}
```

WorkItem 班级

以下 Java 代码代表该WorkItem类。

```
 package com.example.tags;

 public class WorkItem {

 private String key;
 private String name;
 private String confidence ;

public void setKey (String key) {
```

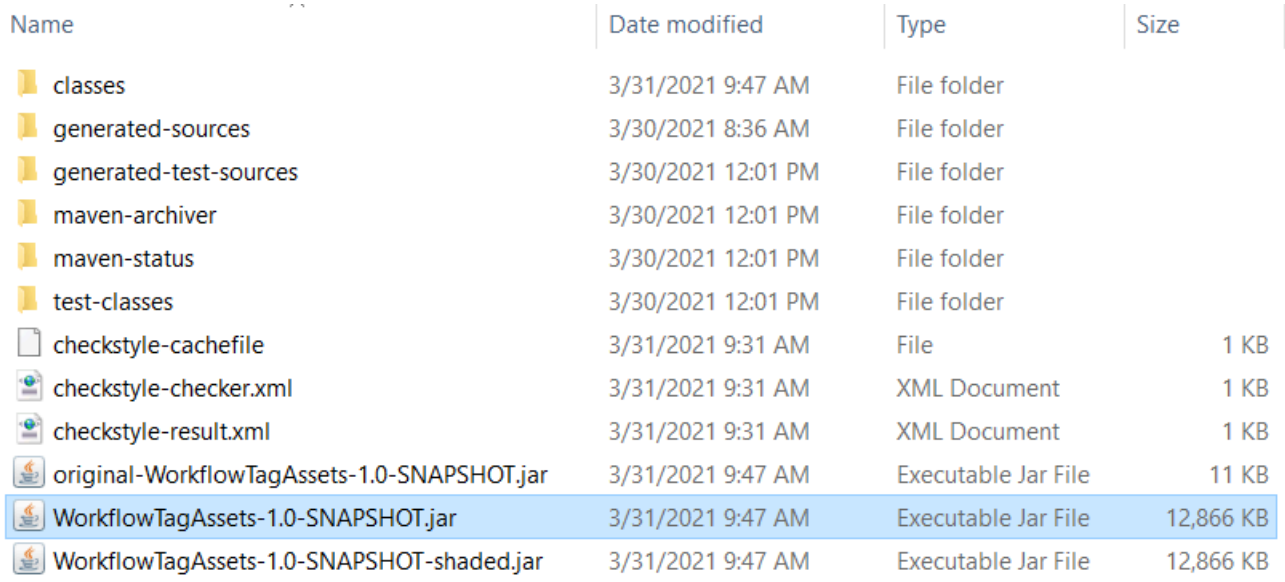
```
    this.key = key;
}

public String getKey() {
    return this.key;
}

public void setName (String name) {
    this.name = name;
}

public String getName() {
    return this.name;
}

public void setConfidence (String confidence) {
    this.confidence = confidence;
}

public String getConfidence() {
    return this.confidence;
}

}
```

打包项目

使用以下 Maven 命令将项目打包成 .jar (JAR) 文件。

```
mvn package
```

JAR 文件位于目标文件夹（这是项目文件夹的子文件夹）。



Name	Date modified	Type	Size
classes	3/31/2021 9:47 AM	File folder	
generated-sources	3/30/2021 8:36 AM	File folder	
generated-test-sources	3/30/2021 12:01 PM	File folder	
maven-archiver	3/30/2021 12:01 PM	File folder	
maven-status	3/30/2021 12:01 PM	File folder	
test-classes	3/30/2021 12:01 PM	File folder	
checkstyle-cachefile	3/31/2021 9:31 AM	File	1 KB
checkstyle-checker.xml	3/31/2021 9:31 AM	XML Document	1 KB
checkstyle-result.xml	3/31/2021 9:31 AM	XML Document	1 KB
original-WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	11 KB
WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB
WorkflowTagAssets-1.0-SNAPSHOT-shaded.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB

> **Note**
>
> 请注意项目的 POM 文件maven-shade-plugin中使用了。此插件负责创建包含所需依赖项的 JAR。如果您尝试在没有此插件的情况下打包项目，则所需的依赖项不包含在 JAR 文件中，您将遇到. ClassNotFoundException

部署 Lambda 函数

1. 打开 Lambda 控制台。
2. 选择创建函数。
3. 选择从头开始创作。
4. 在基本信息部分，输入 cron 作为名称。
5. 在运行时系统中，选择 Java 8。
6. 选择使用现有角色，然后选择 lambda 支持（您创建的 IAM 角色）。
7. 选择创建函数。
8. 对于代码输入种类，选择上传 .zip 文件或 .jar 文件。
9. 选择上传，然后浏览到您创建的 JAR 文件。
10. 对于处理程序，输入函数的完全限定名称，例如 com.example.tags.Handler:handleRequest（com.example.tags 指定程序包，处理程序是后跟 :: 和方法名称的类）。

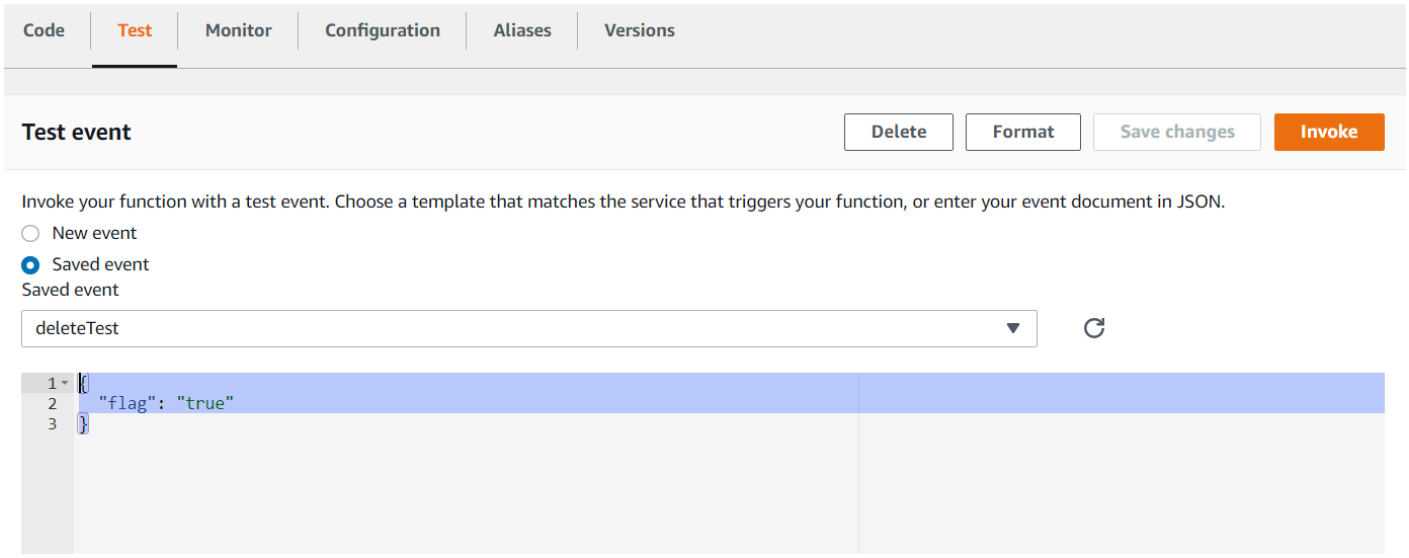
11选择保存。

测试 Lambda 方法

在教程的这一部分，您可以测试 Lambda 函数。

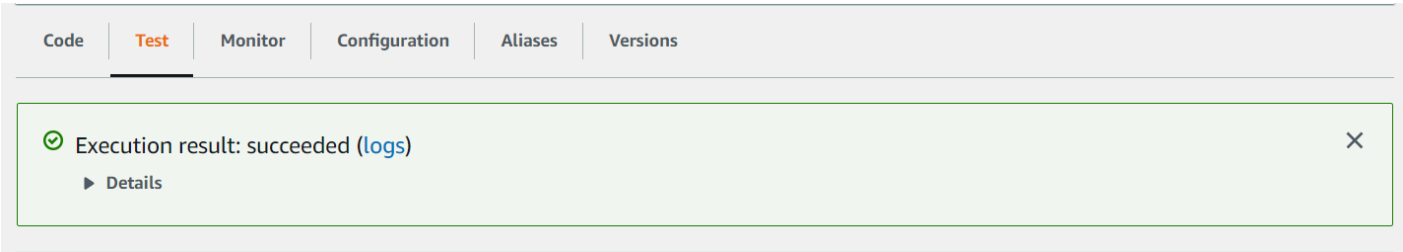
1. 在 Lambda 控制台中，单击测试选项卡，然后输入以下 JSON。

```
                {
"flag": "true"
 }
```



> **Note**
>
> 传递 true 会标记数字资产，传递 false 会删除标签。

2. 选择调用按钮。调用 Lambda 函数后，您会看到一条成功消息。



恭喜，您创建了一个自动将标签应用于 Amazon S3 存储桶中的数字资产的 AWS Lambda 函数。如本教程开头所述，请务必在学习本教程时终止您创建的所有资源，以确保系统不会向您收费。

有关更多 AWS 多服务示例，请参阅AWS 文档 SDK 示例 GitHub 存储库。

创建 AWS 视频分析器应用程序

您可以使用适用于 Java 的 AWS SDK 版本 2 创建用于分析视频以进行标签检测的 Java Web 应用程序。本 AWS 教程中创建的应用程序允许您将视频（MP4 文件）上传到 Amazon S3 存储桶。然后，该应用程序使用 Amazon Rekognition 服务来分析视频。结果用于填充数据模型，然后使用 Amazon Simple Email Service 生成报告并通过电子邮件发送给特定用户。

下图显示了应用程序完成视频分析后生成的报告。下表中的各列显示“年龄范围”、“胡子”、“眼镜”和“眼睛睁开”，以及不同属性预测的置信度值。

	Age Range	Beard	Eye glasses	Eyes open
2				
3	AgeRange(Low=38, High=56)	Beard(Value=false, Confidence=83.07253)	Eyeglasses(Value=true, Confidence=55.965977)	EyeOpen(Value=true, Confidence=94.691696)
4	AgeRange(Low=36, High=52)	Beard(Value=true, Confidence=50.721912)	Eyeglasses(Value=false, Confidence=63.886036)	EyeOpen(Value=true, Confidence=95.906364)
5	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=58.38352)	Eyeglasses(Value=false, Confidence=96.39576)	EyeOpen(Value=true, Confidence=53.580643)
6	AgeRange(Low=49, High=67)	Beard(Value=false, Confidence=81.41662)	Eyeglasses(Value=true, Confidence=65.28722)	EyeOpen(Value=true, Confidence=95.11523)
7	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=61.533833)	Eyeglasses(Value=false, Confidence=97.51163)	EyeOpen(Value=true, Confidence=82.21834)
8	AgeRange(Low=29, High=45)	Beard(Value=false, Confidence=74.22591)	Eyeglasses(Value=true, Confidence=64.906685)	EyeOpen(Value=true, Confidence=98.48175)
9	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=65.9394)	Eyeglasses(Value=false, Confidence=94.14824)	EyeOpen(Value=true, Confidence=94.857346)
10	AgeRange(Low=44, High=62)	Beard(Value=true, Confidence=78.648)	Eyeglasses(Value=true, Confidence=65.83134)	EyeOpen(Value=true, Confidence=98.538666)
11				

在本教程中，您将创建一个调用各种 AWS 服务的 Spring Boot 应用程序。Spring Boot APIs 用于构建模型、不同的视图和控制器。有关更多信息，请参阅 Spring Boot。

此服务使用以下 AWS 服务：

- Amazon Rekognition
- Amazon S3
- Amazon SES
- AWS Elastic Beanstalk

本教程中包含的 AWS 服务包含在 AWS 免费套餐中。我们建议您在使用完本教程中创建的所有资源后将其终止，以免产生费用。

先决条件

在开始之前，您需要完成设置 AWS SDK for Java 中的步骤。然后确保执行以下操作：

- Java 1.8 JDK。

- Maven 3.6 或更高版本。
- 一个名为 video[somevalue] 的 Amazon S3 存储桶。请务必在您的 Amazon S3 Java 代码中使用此存储桶名称。有关更多信息，请参阅创建存储桶。
- IAM 角色。要创建的VideoDetectFaces类需要这个。有关更多信息，请参阅配置 Amazon Rekognition Video。
- 有效的 Amazon SNS 主题。要创建的VideoDetectFaces类需要这个。有关更多信息，请参阅配置 Amazon Rekognition Video。

过程

在本教程的过程中，您将执行以下操作：

1. 创建项目
2. 将 POM 依赖项添加到您的项目中
3. 创建 Java 类
4. 创建 HTML 文件
5. 创建脚本文件
6. 将项目打包到 JAR 文件中
7. 将应用程序部署到 AWS Elastic Beanstalk

要继续学习本教程，请按照AWS 文档 SDK 示例 GitHub 存储库中的详细说明进行操作。

创建 Amazon Rekognition Lambda 函数

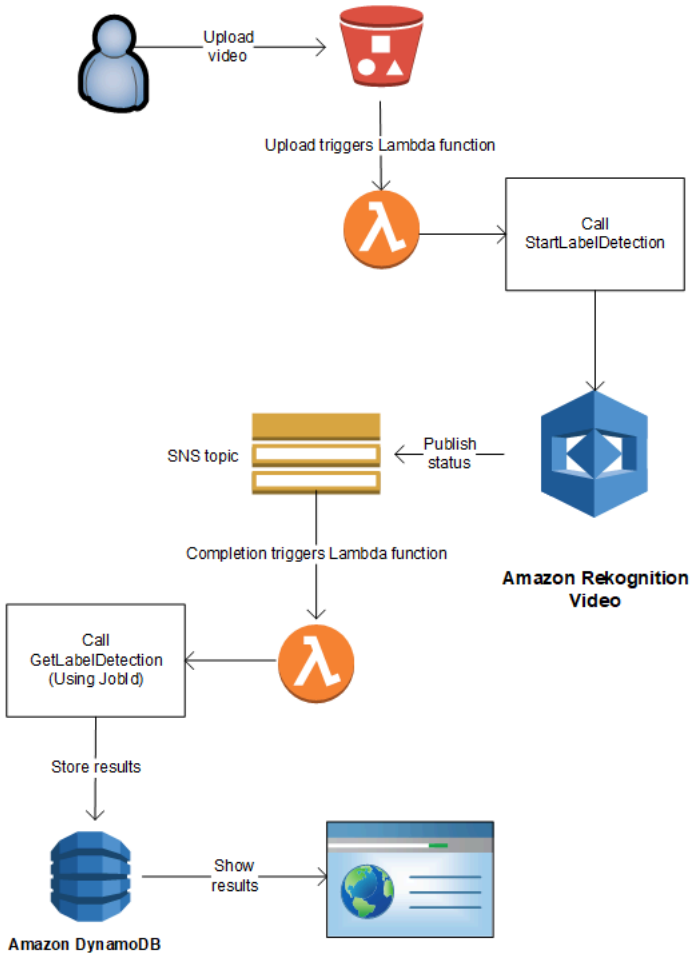
本教程介绍如何使用 Java Lambda 函数获取标签检测的视频分析操作的结果。

> **Note**
>
> 本教程使用适用于 Java 的 AWS SDK 1.x。有关使用 Rekognition 和适用于 AWS Java 的 SDK 版本 2 的教程，请参阅AWS 文档 SDK 示例存储库。 GitHub

您可以将 Lambda 函数与 Amazon Rekognition Video 操作一起使用。例如，下图显示了一个网站，该网站在视频上传到 Amazon S3 存储桶时使用 Lambda 函数自动开始分析视频。触发 Lambda 函

数时，它会调用StartLabelDetection以开始检测上传的视频中的标签。有关使用 Lambda 处理来自 Amazon S3 存储桶的事件通知的信息，请参阅将 AWS Lambda 与 Amazon S3 事件结合使用。

另一个 Lambda 函数在分析完成状态发送到已注册的 Amazon SNS 主题时触发。第二个 Lambda 函数调用GetLabelDetection以获取分析结果。结果随后被存储在数据库中以做好在网页上显示的准备。此第二个 lambda 函数是本教程的重点。



在本教程中，Lambda 函数在 Amazon Rekognition Video 将视频分析的完成状态发送到已注册的 Amazon SNS 主题时触发。它随后通过调用 GetLabelDetection 来收集视频分析结果。出于演示目的，本教程将标签检测结果写入 CloudWatch 日志。在您的应用程序的 Lambda 函数中，您应该存储分析结果以供稍后使用。例如，您可以使用 Amazon DynamoDB 来保存分析结果。有关更多信息，请参阅使用 DynamoDB。

以下步骤将说明如何：

- 创建 Amazon SNS 主题并设置权限。
- 使用创建 Lambda 函数 AWS Management Console 并将其订阅亚马逊 SNS 主题。

- 使用 AWS Management Console配置 Lambda 函数。
- 向 AWS Toolkit for Eclipse 项目添加示例代码并将其上传到 Lambda 函数。
- 使用 AWS CLI测试 Lambda 函数。

> **Note**
>
> 在整个教程中使用相同的 AWS 区域。

先决条件

本教程假定您熟悉 AWS Toolkit for Eclipse。有关更多信息，请参阅 AWS Toolkit for Eclipse。

创建 SNS 主题

Amazon Rekognition Video 视频分析操作的完成状态将发送到 Amazon SNS 主题。此过程将创建 Amazon SNS 主题和 IAM 服务角色（向 Amazon Rekognition Video 授予对您的 Amazon SNS 主题的访问权限）。有关更多信息，请参阅 调用 Amazon Rekognition Video 操作。

创建 Amazon SNS 主题

1. 如果您尚未创建 IAM 服务角色，请创建一个该角色来为 Amazon Rekognition Video 提供对 Amazon SNS 主题的访问权限。记下 Amazon 资源名称（ARN）。有关更多信息，请参阅 提供对多个 Amazon SNS 主题的访问权限。
2. 使用 Amazon SNS 控制台创建 Amazon SNS 主题。您只需要指定主题名称即可。在主题名称前面加上。AmazonRekognition记录主题 ARN。

创建 Lambda 函数

您可使用 AWS Management Console创建 Lambda 函数。然后使用 AWS Toolkit for Eclipse 项目将 Lambda 函数程序包上传到 AWS Lambda。也可以使用 AWS Toolkit for Eclipse创建 Lambda 函数。有关更多信息，请参阅教程：如何创建、上传和调用 AWS Lambda 函数。

创建 Lambda 函数

1. 登录 AWS 管理控制台，然后在上打开 AWS Lambda 控制台。https://console.aws.amazon.com/lambda/

2. 选择创建函数。
3. 选择从头开始创作。
4. 对于函数名称，键入函数的名称。
5. 在运行时系统中，选择 Java 8。
6. 选择选择或创建执行角色。
7. 在执行角色中，选择创建具有基本 Lambda 权限的新角色。
8. 记下基本信息部分底部显示的新角色的名称。
9. 选择创建函数。

配置 Lambda 函数

在创建 Lambda 函数后，您可以将其配置为由您在 创建 SNS 主题 中创建的 Amazon SNS 主题触发。您还可以调整 Lambda 函数的内存要求和超时期限。

配置 Lambda 函数

1. 在函数代码中，为处理程序键入 `com.amazonaws.lambda.demo.JobCompletionHandler`。
2. 在基本设置中，选择编辑。此时将显示编辑基本设置对话框。
   a. 为内存选择 1024。
   b. 对于超时，选择 10 秒。
   c. 选择保存。
3. 在设计器中，选择 + 添加触发器。此时将显示“添加触发器”对话框。
4. 在触发器配置中，选择 SNS。

   在 SNS 主题中，选择您在 创建 SNS 主题 中创建的 Amazon SNS 主题。
5. 选择启用触发器。
6. 要添加触发器，请选择添加。
7. 选择保存以保存 Lambda 函数。

配置 IAM Lambda 角色

要调用 Amazon Rekognition Video 操作，您需要将 AmazonRekognitionFullAccessAWS 托管策略添加到 IAM Lambda 角色中。启动操作（例如 StartLabelDetection）还需要 Amazon Rekognition Video 用于访问亚马逊 SNS 主题的 IAM 服务角色的传递角色权限。

配置角色

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。
3. 在列表中，选择您在 创建 Lambda 函数中创建的执行角色的名称。
4. 选择 Permissions（权限）选项卡。
5. 选择附加策略。
6. AmazonRekognitionFullAccess从策略列表中选择。
7. 选择附加策略。
8. 再次选择此执行角色。
9. 选择添加内联策略。
10. 选择 JSON 选项卡。
11. 将现有策略替换为以下策略。将 `servicerole` 替换为您在 创建 SNS 主题 中创建的 IAM 服务角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "mysid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:servicerole"
        }
    ]
}
```

12. 选择查看策略。
13. 在名称* 中键入策略的名称。
14. 选择创建策略。

创建 AWS Toolkit for Eclipse Lambda 项目

触发 Lambda 函数时，以下代码会从 Amazon SNS 主题中获取完成状态，并GetLabelDetection调用以获取分析结果。将检测到的标签计数和检测到的标签列表写入 CloudWatch 日志。您的 Lambda 函数应该存储视频分析结果以供稍后使用。

创建 AWS Toolkit for Eclipse Lambda 项目

1. 创建一个 AWS Toolkit for EclipseAWS Lambda 项目。
   - 对于项目名称:，键入您选择的项目名称。
   - 在 “类名:” 中，输入JobCompletionHandler。
   - 对于输入类型:，选择 SNS 事件。
   - 使其他字段保持不变。
2. 在 Eclipse 项目资源管理器中，打开生成的 Lambda 处理程序方法 (JobCompletionHandler.java)，然后将内容替换为以下内容：

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.lambda.demo;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import java.util.List;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
public class JobCompletionHandler implements RequestHandler<SNSEvent, String> {

   @Override
   public String handleRequest(SNSEvent event, Context context) {

      String message = event.getRecords().get(0).getSNS().getMessage();
      LambdaLogger logger = context.getLogger();

      // Parse SNS event for analysis results. Log results
      try {
         ObjectMapper operationResultMapper = new ObjectMapper();
         JsonNode jsonResultTree = operationResultMapper.readTree(message);
         logger.log("Rekognition Video Operation:========================");
         logger.log("Job id: " + jsonResultTree.get("JobId"));
         logger.log("Status : " + jsonResultTree.get("Status"));
         logger.log("Job tag : " + jsonResultTree.get("JobTag"));
         logger.log("Operation : " + jsonResultTree.get("API"));

         if (jsonResultTree.get("API").asText().equals("StartLabelDetection")) {

            if (jsonResultTree.get("Status").asText().equals("SUCCEEDED")){
               GetResultsLabels(jsonResultTree.get("JobId").asText(), context);
            }
            else{
               String errorMessage = "Video analysis failed for job "
                     + jsonResultTree.get("JobId")
                     + "State " + jsonResultTree.get("Status");
               throw new Exception(errorMessage);
            }

         } else
            logger.log("Operation not StartLabelDetection");

      } catch (Exception e) {
         logger.log("Error: " + e.getMessage());
         throw new RuntimeException (e);


      }

      return message;
   }

   void GetResultsLabels(String startJobId, Context context) throws Exception {
```

```
     LambdaLogger logger = context.getLogger();

     AmazonRekognition rek =
AmazonRekognitionClientBuilder.standard().withRegion(Regions.US_EAST_1).build();

     int maxResults = 1000;
     String paginationToken = null;
     GetLabelDetectionResult labelDetectionResult = null;
     String labels = "";
     Integer labelsCount = 0;
     String label = "";
     String currentLabel = "";

     //Get label detection results and log them.
     do {

        GetLabelDetectionRequest labelDetectionRequest = new
GetLabelDetectionRequest().withJobId(startJobId)

.withSortBy(LabelDetectionSortBy.NAME).withMaxResults(maxResults).withNextToken(pagination

        labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

        paginationToken = labelDetectionResult.getNextToken();
        VideoMetadata videoMetaData = labelDetectionResult.getVideoMetadata();

        // Add labels to log
        List<LabelDetection> detectedLabels = labelDetectionResult.getLabels();

        for (LabelDetection detectedLabel : detectedLabels) {
           label = detectedLabel.getLabel().getName();
           if (label.equals(currentLabel)) {
              continue;
           }
           labels = labels + label + " / ";
           currentLabel = label;
           labelsCount++;

        }
     } while (labelDetectionResult != null &&
labelDetectionResult.getNextToken() != null);

     logger.log("Total number of labels : " + labelsCount);
```

```
        logger.log("labels : " + labels);

    }


}
```

3. Rekognition 命名空间未被解析。纠正方法：
   - 将鼠标指针停留在行 `import com.amazonaws.services.rekognition.AmazonRekognition;` 的带下划线的部分。
   - 选择修复项目设置...。
   - 选择最新版本的 Amazon Rekognition 存档。
   - 选择确定以将存档保存到项目。
4. 保存该文件。
5. 在 Eclipse 代码窗口中右键单击，选择 AWS Lambda，然后选择将函数上传到 AWS Lambda。
6. 在选择目标 Lambda 函数页面上，选择要使用的 AWS 区域。
7. 选择选择现有 Lambda 函数，然后选择您在 创建 Lambda 函数 中创建的 Lambda 函数。
8. 选择下一步。此时将显示函数配置对话框。
9. 在 IAM 角色中，选择您在 创建 Lambda 函数 中创建的 IAM 角色。
10. 选择完成，Lambda 函数将上传到 AWS。

测试 Lambda 函数

使用以下 AWS CLI 命令通过启动视频的标签检测分析来测试 Lambda 函数。分析完成后，将触发 Lambda 函数。通过检查 CloudWatch 日志来确认分析成功。

测试 Lambda 函数

1. 将 MOV 或 MPEG-4 格式的视频文件上传到您的 S3 存储桶。对于测试，请上传时长不超过 30 秒的视频。

   有关说明，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到 Amazon S3。

2. 运行以下 AWS CLI 命令开始检测视频中的标签。

```
aws rekognition start-label-detection --video
 "S3Object={Bucket="bucketname",Name="videofile"}" \
--notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
--region Region
```

更新以下值：

- 将 bucketname 和 videofile 更改为您要在其中检测标签的视频的 Amazon S3 存储桶名称和文件名。
- 将 TopicARN 更改为您在 创建 SNS 主题 中创建的 Amazon SNS 主题的 ARN。
- 将 RoleARN 更改为您在 创建 SNS 主题 中创建的 IAM 角色的 ARN。
- 切换Region到您正在使用的 AWS 区域。

3. 记下响应中 JobId 的值。该响应看上去与以下 JSON 示例类似。

```
{
    "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
}
```

4. 打开 https://console.aws.amazon.com/cloudwatch/ 控制台。
5. 当分析完成时，Lambda 函数的日志条目将显示在日志组中。
6. 选择 Lambda 函数以查看日志流。
7. 选择最新日志流以查看由 Lambda 函数制作的日志条目。如果操作成功，则显示类似于以下输出，其中显示了视频识别操作的详细信息，包括作业 ID、操作类型 StartLabelDetection “” 以及检测到的标签类别列表，例如瓶子、服装、人群和食物：

Time (UTC +00:00)	Message
2018-02-28	
19:48:01	START RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Version: $LATEST
19:48:02	Rekognition Video Operation:==========
19:48:02	Job id: "9c7c3b1403a375a044c6dbe793d5c78d06014ee16f5efde083ad654b06f6c59a"
19:48:02	Status : "SUCCEEDED"
19:48:02	Job tag : null
19:48:02	Operation : "StartLabelDetection"
19:48:09	Total number of labels : 29
19:48:09	labels : Audience / Badge / Bottle / Clothing / Coat / Crowd / Electric Guitar / Flora / Food / Guitar / Hu
19:48:09	Result: {}
19:48:09	END RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b
19:48:09	REPORT RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Duration: 8036.70 ms Billed Duration:

任务 ID 的值应与您在步骤 3 中记下的 `JobId` 的值匹配。

使用 Amazon Rekognition 进行身份验证

Amazon Rekognition 为用户提供了多种操作，可以轻松创建身份验证系统。Amazon Rekognition 允许用户检测图像中的人脸，然后通过比较人脸数据将任何检测到的人脸与其他人脸进行比较。这些人脸数据存储在名为“集合”的服务器端容器中。通过利用 Amazon Rekognition 的人脸检测、人脸比较和集合管理操作，您可以创建带有身份验证解决方案的应用程序。

本教程将演示创建需要身份验证的应用程序的两个常见工作流程。

第一个工作流程涉及在集合中注册新用户。第二个工作流程涉及搜索现有集合，以便登录回归用户。

在本教程中，您将使用 AWS SDK for Python。您还可以查看 AWS 文档 SDK 示例存储库，了解更多 GitHub P ython 教程。

主题

- 先决条件
- 创建集合
- 新用户注册
- 现有用户登录

先决条件

在开始本教程之前，你需要安装 Python 并完成设置 Python AWS 开发工具包所需的步骤。除此之外，请确保您：

- 已创建 AWS 账户和 IAM 角色
- 已安装 Python SDK (Boto3)
- 正确配置您的 AWS 访问凭证
- 已创建 Amazon Simple Storage Service 存储桶，并上传一张您希望用作身份验证的 ID 的图像。
- 已选择第二张图像作为身份验证的目标图像。

创建集合

在集合中注册新用户或搜索用户集合之前，必须先有一个集合可供使用。Amazon Rekognition 集合是一个服务器端容器，用于存储有关检测到的人脸的信息。

创建集合

首先，您将编写一个函数，用于创建供应用程序使用的集合。Amazon Rekognition 存储有关在名为集合的服务器端容器中检测到的人脸的信息。您可以搜索存储在集合中的人脸信息以查找已知人脸。要存储人脸信息，您首先需要使用 `CreateCollection` 操作创建集合。

1. 请为要创建的集合选择名称。在以下代码中，将 `collection_id` 的值替换为您要创建的集合的名称，并将`region`的值替换为用户凭证中定义的区域名称。您可以使用`Tags`参数将您想要的任何标签应用到集合中，尽管这不是必需的。`CreateCollection` 操作将返回有关您创建的集合的信息，包括该集合的 Arn。记下运行代码后收到的 Arn。

```
import boto3

def create_collection(collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    # Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId=collection_id,
    Tags={"SampleKey1":"SampleValue1"})
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

collection_id = 'collection-id-name'
region = "region-name"
create_collection(collection_id, region)
```

2. 保存并运行代码。复制集合 Arn。

现在 Rekognition 集合已经创建，您可以在该集合中存储人脸信息和标识符。您还可以将人脸与存储的信息进行比较以进行验证。

新用户注册

您需要能够注册新用户并将其信息添加到集合中。注册新用户的过程通常涉及以下步骤：

调用 **DetectFaces** 操作

编写代码以通过 DetectFaces 操作检查人脸图像的质量。您将使用 DetectFaces 操作来确定摄像头拍摄的图像是否适合由 SearchFacesByImage 操作处理。图像应仅包含一张人脸。您将为 DetectFaces 操作提供本地输入图像文件，并接收图像中检测到的人脸的详细信息。以下示例代码为 DetectFaces 提供输入图像，然后检查图像中是否仅检测到一张人脸。

1. 在以下代码示例中，将photo替换为要在其中检测人脸的目标图像的名称。您还需要将region的值替换为与您的账户关联的地区名称。

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    Attributes=['ALL'])

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])
```

```
photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))

if face_count == 1:
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. 保存并运行正在进行的代码。

调用 **CompareFaces** 操作

您的应用程序需要能够在集合中注册新用户并确认回归用户的身份。您将首先创建用于注册新用户的函数。首先，您将使用该CompareFaces操作来比较本地input/target image of the user and a ID/stored 图像。如果在两张图片中检测到的人脸相符，则可以在集合中进行搜索，以查看用户是否已在其中注册。

首先编写一个函数，将输入图像与您存储在 Amazon S3 存储桶中的 ID 图像进行比较。在以下代码示例中，您需要自行提供输入图像，该图像应在使用某种形式的活跃度检测器后拍摄。您还需要传递存储在 Amazon S3 存储桶中的图像的名称。

1. 将bucket的值替换为包含源文件的 Amazon S3 存储桶的名称。您还需要将source_file的值替换为您正在使用的源图像的名称。将target_file的值替换为您提供的目标文件的名称。将region的值替换为用户凭证中定义的region的名称。

   您还需要使用 similarityThreshold 参数指定响应中返回的匹配结果的最低置信度。只有当置信度高于此阈值时，才会在 FaceMatches 数组中返回检测到的人脸。您选择的 similarityThreshold 应反映特定使用案例的性质。任何涉及关键安全应用的用例都应使用 99 作为所选阈值。

   ```
   import boto3

   def compare_faces(bucket, sourceFile, targetFile, region):
       client = boto3.client('rekognition', region_name=region)

       imageTarget = open(targetFile, 'rb')

       response = client.compare_faces(SimilarityThreshold=99,
   ```

```
                                    SourceImage={'S3Object':
{'Bucket':bucket,'Name':sourceFile}},
                                    TargetImage={'Bytes': imageTarget.read()})

    for faceMatch in response['FaceMatches']:
        position = faceMatch['Face']['BoundingBox']
        similarity = str(faceMatch['Similarity'])
        print('The face at ' +
              str(position['Left']) + ' ' +
              str(position['Top']) +
              ' matches with ' + similarity + '% confidence')

    imageTarget.close()
    return len(response['FaceMatches'])

bucket = 'bucket-name'
source_file = 'source-file-name'
target_file = 'target-file-name'
region = "region-name"
face_matches = compare_faces(bucket, source_file, target_file, region)
print("Face matches: " + str(face_matches))

if str(face_matches) == "1":
    print("Face match found.")
else:
    print("No face match found.")
```

2. 保存并运行正在进行的代码。

   系统将返回一个响应对象，其中包含有关匹配人脸和置信度的信息。

调用 **SearchFacesByImage** 操作

如果 CompareFaces 操作的置信度高于您选择的 SimilarityThreshold，则需要在集合中搜索可能与输入图像匹配的人脸。如果在您的集合中找到了匹配项，则表示该用户可能已经在集合中注册，因此无需在您的集合中注册新用户。如果没有匹配项，则可以在集合中注册新用户。

1. 首先编写将调用 SearchFacesByImage 操作的代码。该操作将以本地图像文件作为参数，然后在您的Collection搜索中搜索与所提供图像中检测到的最大面孔匹配的人脸。

在以下代码示例中，将collectionId 的值更改为要搜索的集合。将region的值替换为与您的账户关联的区域名称。您还需要将photo的值替换为输入文件的名称。您还需要通过将threshold的值替换为选定的百分位数来指定相似度阈值。

```
import boto3

collectionId = 'collection-id-name'
region = "region-name"
photo = 'photo-name'
threshold = 99
maxFaces = 1
client = boto3.client('rekognition', region_name=region)

# input image should be local file here, not s3 file
with open(photo, 'rb') as image:
    response = client.search_faces_by_image(CollectionId=collectionId,
    Image={'Bytes': image.read()},
    FaceMatchThreshold=threshold, MaxFaces=maxFaces)

faceMatches = response['FaceMatches']
print(faceMatches)

for match in faceMatches:
    print('FaceId:' + match['Face']['FaceId'])
    print('ImageId:' + match['Face']['ImageId'])
    print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
    print('Confidence: ' + str(match['Face']['Confidence']))
```

2. 保存并运行正在进行的代码。如果有匹配项，则表示图像中识别出的人已经是集合的一部分，无需继续执行下一步操作。在这种情况下，您可以只允许用户访问应用程序。

调用 **IndexFaces** 操作

假设在您搜索的集合中未找到匹配项，则需要将该用户的人脸添加到您的集合中。您可以通过调用 IndexFaces 操作来完成操作。当您致电时 IndexFaces，Amazon Rekognition 会提取您输入图像中识别出的人脸的面部特征，并将数据存储在指定的集合中。

1. 首先编写代码以调用 IndexFaces。将的image值替换为要用作 IndexFaces 操作输入图像的本地文件的名称。您还需要将photo_name的值替换为输入图像所需的名称。请确保将collection_id 的值替换为您之前创建的集合的 ID。接下来，将region的值替换为与您的账

户关联的区域名称。您还需要为 MaxFaces 输入参数指定一个值，该参数定义了图像中应编制索引的最大人脸数。该参数的默认值为 1。

```
import boto3

def add_faces_to_collection(target_file, photo, collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'Bytes': imageTarget.read()},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])
    print(response)

    print('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
        print('  Face ID: ' + faceRecord['Face']['FaceId'])
        print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))
        print('  Image ID: {}'.format(faceRecord['Face']['ImageId']))
        print('  External Image ID: {}'.format(faceRecord['Face']
['ExternalImageId']))
        print('  Confidence: {}'.format(faceRecord['Face']['Confidence']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])

image = 'image-file-name'
collection_id = 'collection-id-name'
photo_name = 'desired-image-name'
region = "region-name"

indexed_faces_count = add_faces_to_collection(image, photo_name, collection_id,
 region)
```

```
print("Faces indexed count: " + str(indexed_faces_count))
```

2. 保存并运行正在进行的代码。确定是否要保存 IndexFaces 操作返回的任何数据，例如分配给图像中人物的 FaceID。下一节将探讨如何保存这些数据。在继续操作之前，请复制返回的 FaceId、ImageId 和Confidence值。

将图像和 FaceID 数据存储在 Amazon S3 和 Amazon DynamoDB 中

获取输入图像的人脸 ID 后，可以将图像数据保存在 Amazon S3 中，而人脸数据和图像 URL 可以输入到 DynamoDB 等数据库中。

1. 编写代码，将输入图像上传到 Amazon S3 数据库。在下面的代码示例中，将bucket的值替换为您要将文件上传到的存储桶的名称，然后将file_name的值替换为您要存储在 Amazon S3 存储桶中的本地文件的名称。提供一个用于识别 Amazon S3 存储桶中文件的密钥名称，方法是将key_name的值替换为您想要为图像文件命名的名称。您要上传的文件与之前的代码示例中定义的文件相同，也就是您使用的输入文件 IndexFaces。最后，将region的值替换为与您的账户关联的区域名称。

```
import boto3
import logging
from botocore.exceptions import ClientError

# store local file in S3 bucket
bucket = "amzn-s3-demo-bucket"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
s3 = boto3.client('s3', region_name=region)
# Upload the file
try:
    response = s3.upload_file(file_name, bucket, key_name)
    print("File upload successful!")
except ClientError as e:
    logging.error(e)
```

2. 保存并运行后续代码示例，将您的输入图像上传到 Amazon Amazon S3。
3. 您还需要将返回的人脸 ID 保存到数据库中。这可以通过创建 DynamoDB 数据库表，然后将人脸 ID 上传到该表来完成。以下代码示例将创建 DynamoDB 表。请注意，您只需要运行一次创建此表的代码。在以下代码中，将region的值替换为与您的账户关联的区域的值。您还需要将database_name的值替换为您想要给 DynamoDB 表的名称。

```
import boto3

# Create DynamoDB database with image URL and face data, face ID

def create_dynamodb_table(table_name, region):
    dynamodb = boto3.client("dynamodb", region_name=region)

    table = dynamodb.create_table(
        TableName=table_name,
        KeySchema=[{
                'AttributeName': 'FaceID', 'KeyType': 'HASH'  # Partition key
                },],
            AttributeDefinitions=[
            {
                'AttributeName': 'FaceID', 'AttributeType': 'S'  }, ],
                ProvisionedThroughput={
            'ReadCapacityUnits': 10, 'WriteCapacityUnits': 10  }
    )
    print(table)
    return table

region = "region-name"
database_name = 'database-name'
dynamodb_table = create_dynamodb_table(database_name, region)
print("Table status:", dynamodb_table)
```

4. 保存并运行后续代码以创建您的表。
5. 创建表后，您可以将返回的内容上传 FaceId 到表中。为此，您需要使用表函数与表建立连接，然后使用put_item函数上传数据。

   在以下代码示例中，将bucket的值替换为包含您上传到 Amazon S3 的输入图像的存储桶名称。您还需要将 file_name的值替换为您上传到 Amazon S3 存储桶的输入文件的名称，并将key_name的值替换为您之前用于识别输入文件的密钥。最后，将region的值替换为与您的账户关联的区域名称。这些值应与步骤 1 中提供的值相匹配。

   将分配给人脸的 FaceId ImageId、和置信度值AddDBEntry存储在集合中。在下面的函数中提供您在后续 IndexFaces 节的步骤 2 中保存的值。

```
import boto3
from pprint import pprint
from decimal import Decimal
```

```
import json

# The local file that was stored in S3 bucket
bucket = "amzn-s3-demo-bucket"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
# Get URL of file
file_url = "https://s3.amazonaws.com/{}/{}".format(bucket, key_name)
print(file_url)

# upload face-id, face info, and image url
def AddDBEntry(file_name, file_url, face_id, image_id, confidence):
    dynamodb = boto3.resource('dynamodb', region_name=region)
    table = dynamodb.Table('FacesDB-4')
    response = table.put_item(
       Item={
            'ExternalImageID': file_name,
            'ImageURL': file_url,
            'FaceID': face_id,
            'ImageID': image_id,
            'Confidence': json.loads(json.dumps(confidence), parse_float=Decimal)
       }
    )
    return response

# Mock values for face ID, image ID, and confidence - replace them with actual
 values from your collection results
dynamodb_resp = AddDBEntry(file_name, file_url, "FACE-ID-HERE",
    "IMAGE-ID-HERE", confidence-here)
print("Database entry successful.")
pprint(dynamodb_resp, sort_dicts=False)
```

6. 保存并运行后续代码示例，将返回的人脸 ID 数据存储在表中。

现有用户登录

在集合中注册用户后，可以使用 SearchFacesByImage 操作在返回时对其进行身份验证。您需要获取输入图像，然后使用 DetectFaces 检查输入图像的质量。这决定了在运行 SearchFacesbyImage 操作之前是否使用了合适的图像。

致电 DetectFaces 行动

1. 您将使用 DetectFaces 操作来检查人脸图像的质量，并确定摄像头拍摄的图像是否适合由 SearchFacesByImage 操作处理。输入图像应仅包含一张人脸。以下代码示例获取输入图像并将其提供给 DetectFaces 操作。

   在以下代码示例中，将photo的值替换为本地目标图像的名称，并将region的值替换为与您的账户关联的区域名称。

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    Attributes=['ALL'])

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')

        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))

if face_count == 1:
```

```
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. 保存并运行代码。

致电 SearchFacesByImage 行动

1. 编写代码，使用 SearchFacesByImage 将检测到的人脸与集合中的人脸进行比较。您将使用后续“新用户注册”部分中显示的代码，并为 SearchFacesByImage 操作提供输入图像。

   在以下代码示例中，将 collectionId 的值更改为要搜索的集合。您还要将bucket的值更改为 Amazon S3 存储桶的名称，将fileName的值更改为该存储桶中的图像文件。将region的值替换为与您的账户关联的区域名称。您还需要通过将threshold的值替换为选定的百分位数来指定相似度阈值。

   ```
   import boto3

   bucket = 'amzn-s3-demo-bucket'
   collectionId = 'collection-id-name'
   region = "region-name"
   fileName = 'file-name'
   threshold = 70
   maxFaces = 1
   client = boto3.client('rekognition', region_name=region)

   # input image should be local file here, not s3 file
   with open(fileName, 'rb') as image:
       response = client.search_faces_by_image(CollectionId=collectionId,
       Image={'Bytes': image.read()},
       FaceMatchThreshold=threshold, MaxFaces=maxFaces)
   ```

2. 保存并运行代码。

查看返回的 FaceID 和置信度

现在，您可以 FaceId 通过打印出响应元素（如 “相似度” 和 “ FaceId置信度” 属性）来检查匹配项的信息。

```
faceMatches = response['FaceMatches']
print(faceMatches)
```

```
for match in faceMatches:
    print('FaceId:' + match['Face']['FaceId'])
    print('ImageId:' + match['Face']['ImageId'])
    print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
    print('Confidence: ' + str(match['Face']['Confidence']))
```

使用 Amazon Rekognition（REK）检测用于营销应用程序的标签

本教程将指导您根据上传到网站的图像，构建一个可用于向他人发送电子邮件的示例 Python 应用程序。该示例应用程序旨在通过向用户发送有关旅行优惠的个性化电子邮件（如果照片中识别出地标），从而吸引用户参与社交媒体营销活动。

该解决方案集成了各种 AWS 服务，包括亚马逊 Rekognition、Amazon S3 CloudWatch、DynamoDB 和亚马逊 SES。该应用程序使用 Amazon Rekognition 来检测由 DynamoDB 数据库中识别的账户上传的图像中的标签，然后根据图像中检测到的标签向账户持有人发送一封营销电子邮件。完整的解决方案架构如下：

- 将用户数据存储在 DynamoDB 数据库中。
- 用户将图像数据和元数据（用户账号）上传到 Amazon S3。
- 调用 Lambda 函数DetectLabels来识别和记录上传图像中的标签，在 DynamoDB 中查找用户电子邮件，并向上传图像的用户发送营销电子邮件。
- Lambda 函数的结果已登录以 CloudWatch 供日后查看。

本教程中所有步骤概述如下：

1. 创建和填充 DynamoDB 表。
2. 编写具有日志记录和通知功能的 Lambda 函数。
3. 设置 Lambda 函数和权限。
4. 使用 Amazon S3 上传图像和元数据。
5. 民意调查 CloudWatch 日志。

主题

- 先决条件
- 创建和填充 DynamoDB 表
- 创建具有日志记录和通知功能的 Lambda 函数

- 设置权限和 Lambda 函数
- 使用元数据将图像上传到 Amazon S3
- 来自客户端的轮询 CloudWatch 日志

先决条件

开始本教程之前，您将需要：

- AWS 账户和相应的 IAM 权限。
- 已在开发环境中安装 Python 和 Boto3。
- 对 Lambda、亚马逊 S3、DynamoDB、Amazon SES 和. CloudWatch
- 基本熟悉 Lambda 和 Amazon SES。

创建和填充 DynamoDB 表

在本教程开始时，我们将使用 AccountNumber “” 变量作为主键创建一个 DynamoDB 表来存储客户数据，例如：电子邮件地址、年龄、电话号码和会员状态。我们还将在表中插入一些示例数据。

创建 DynamoDB 表

首先，我们创建 DynamoDB 表。下面介绍了如何使用 Python 和 Boto3 进行设置。我们创建了一个名为 `create_user_table` 的函数并连接到其中的 DynamoDB 资源。在下面的代码示例中，将“region_name”的值替换为您的账户运营所在区域的名称，然后运行代码单元来创建您的表。

```
import boto3
def create_user_table(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.create_table(
        TableName='CustomerDataTable',
        KeySchema=[
            {
                'AttributeName': 'AccountNumber',
                'KeyType': 'HASH'  # Partition key
            },
        ],
        AttributeDefinitions=[
```

```
            {
                'AttributeName': 'AccountNumber',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )

    # Wait until the table exists.
    table.wait_until_exists()
    print("Table status:", table.table_status)

# Create the DynamoDB table.
create_user_table()
```

运行此脚本已设置了一个名为 CustomerDataTable 的 DynamoDB 表，以 AccountNumber 作为主键。

插入示例数据

现在，我们要在表中插入一些示例数据。此示例数据将帮助我们测试应用程序的完整功能。

我们将创建第二个函数，该函数将示例数据添加到我们之前创建的 CustomerDataTable 中。以下代码创建了三个示例条目，每个条目都包括账号、电子邮件地址、年龄、电话号码和成员身份状态。在下面的代码示例中，将 region_name 的值替换为您的账户运营所在区域的名称，然后运行代码单元来创建您的表。如果您想测试应用程序的电子邮件发送部分，请将下面第一个客户条目中 EmailAddress 的值替换为您可以接收电子邮件的电子邮件地址。保存并运行代码。

```
import boto3

def insert_sample_data(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.Table('CustomerDataTable')

    # Sample data
```

```
    customers = [
        {
            'AccountNumber': 'ACC1000',
            'EmailAddress': 'email-for-delivery-here',
            'Age': 30,
            'PhoneNumber': '123-456-7890',
            'MembershipStatus': 'Active'
        },
        {
            'AccountNumber': 'ACC1001',
            'EmailAddress': 'jane.doe@example.com',
            'Age': 25,
            'PhoneNumber': '098-765-4321',
            'MembershipStatus': 'Inactive'
        },
        {
            'AccountNumber': 'ACC1002',
            'EmailAddress': 'pat.candella@example.com',
            'Age': 35,
            'PhoneNumber': '555-555-5555',
            'MembershipStatus': 'Active'
        }
    ]
        # Inserting data
    for customer in customers:
            print(f"Adding customer: {customer['AccountNumber']}")
            table.put_item(Item=customer)

# Insert sample data into DynamoDB
insert_sample_data()
```

设置并填充 DynamoDB 表后，我们现在可以继续将此数据检索集成到 Lambda 函数中。这样，我们的应用程序就可以根据在即将进行的图像处理工作流中识别的账号来检索用户详细信息。

创建具有日志记录和通知功能的 Lambda 函数

现在，我们可以创建一个 Lambda 函数。我们希望确保图像上传时触发的 Lambda 函数可以读取图像数据和账户元数据，并用它在 DynamoDB 表中查找关联的用户数据。这意味着我们不仅需要调用 Amazon Rekognition 的 DetectLabels 函数，还需要一个接收 AccountNumber 并使用它从 DynamoDB 检索关联电子邮件地址的函数。此外，我们还需要一个函数来使用 Amazon SES 向电子邮件地址发送电子邮件。最后，我们使用记录器来记录有关该过程的信息。这些数据将由显示 CloudWatch。

创建 Lambda 函数

下面概述了 Lambda 函数可能如何满足这些要求。在以下代码示例中，如果您使用了 CustomerDataTable 其他名称作为表名，请确保指定了正确的 DynamoDB 表。此外，在“send_marketing_email”函数中，应将“Source”参数的值替换为您具有访问权限的电子邮件地址，该电子邮件地址将用于发送电子邮件。

```
import json
import boto3
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    s3_bucket = event['Records'][0]['s3']['bucket']['name']
    s3_object_key = event['Records'][0]['s3']['object']['key']
    print(s3_bucket)
    print(s3_object_key)

    s3 = boto3.client('s3')
    try:
        s3_response = s3.head_object(Bucket=s3_bucket, Key=s3_object_key)
        account_number = s3_response['Metadata']['account_number']
    except Exception as e:
        logger.error(f"Failed to retrieve object or metadata: {str(e)}")
        raise e  # Optionally re-raise to handle the error upstream or signal a failure

    rekognition = boto3.client('rekognition')
    try:
        labels_response = rekognition.detect_labels(Image={'S3Object': {'Bucket':
 s3_bucket, 'Name': s3_object_key}})
        #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'],
 indent=4)}")
    except Exception as e:
        #logger.info(f"Detected label: {label['Name']}")
        raise e

    #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'], indent=4)}")

    landmark_detected = any(label['Name'] == 'Landmark' and label['Confidence'] > 20
 for label in labels_response['Labels'])
    if landmark_detected:
```

```
        result = notify_user_based_on_landmark(account_number)
        print(result)
        #logger.info(f"Detected label: {label['Name']}")
        #logger.info(f"Notification sent: {result}")
    return {
        'statusCode': 200,
        'body': json.dumps('Process completed successfully!')
    }

def notify_user_based_on_landmark(account_number):
     # Retrieve user data from DynamoDB
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('CustomerDataTable')
    user_info = table.get_item(Key={'AccountNumber': account_number})

    # Send email if user is found
    if 'Item' in user_info:
        send_marketing_email(user_info['Item']['EmailAddress'])
    return "User notified with relevant travel information."

def send_marketing_email(email):
    ses = boto3.client('ses')
    response = ses.send_email(
        Source='your-email@example.com',
        Destination={'ToAddresses': [email]},
        Message={
            'Subject': {'Data': 'Explore New Destinations!'},
            'Body': {
                'Text': {'Data': 'Check out our exclusive travel packages inspired by
 the landmark in your uploaded image!'}
            }
        }
    )
    return f"Email sent to {email} with status {response['ResponseMetadata']
['HTTPStatusCode']}"
    print("succeess")
```

现在我们已经编写了 Lambda 函数，我们需要为 Lambda 配置权限并在管理控制台中创建 Lambda 函数的实例。 AWS

设置权限和 Lambda 函数

创建或更新 IAM 角色

我们必须处理 Lambda 函数的权限，才能使用该函数处理用户上传的任何图像。Lambda 函数需要一个 IAM 角色的作用域，该角色具有允许其与亚马逊 S3、DynamoDB、Amazon SES、Amazon Rekognition 和进行交互的策略。 CloudWatch

为 Lambda 函数配置 IAM 角色：

1. 打开 AWS 管理控制台。
2. 前往“IAM”>“角色”>“创建角色”。

   选择“创建角色”。
3. 选择 Lambda 作为将使用此角色的服务。单击“下一步”。
4. 附加以下策略（请注意，选择这些策略仅用于演示目的，在实际生产环境中，您可能希望仅对需要的策略授予权限。）：

   - 亚马逊 3 ReadOnlyAccess
   - AmazonRekognitionReadOnlyAccess
   - AmazonDynamoDBFull访问权限
   - 亚马逊SESFull访问权限
   - AWSLambda执行
   - AWSLambdaBasicExecutionRole （用于 CloudWatch 记录）

   单击“下一步”。
5. 为角色命名并进行描述，然后通过选择“创建角色”来创建角色。
6. 一旦我们为 Lambda 配置了适当的权限，我们就可以使用管理控制台创建 Lambda 函数的实例。AWS

   在 AWS 管理控制台中，转到 Lambda 服务。
7. 单击“创建函数”。选择“从头开始创作”。
8. 输入一个函数名称并选择 Python 运行时。在“更改默认执行角色”下，选择“使用现有角色”，然后选择您之前创建的 IAM 角色。
9. 现在，您必须在 Lambda 函数选项卡中创建和更新代码。转到名为“lambda_function”的选项卡，然后将那里的代码替换为后续的 Lambda 代码示例。

保存更改并部署更改。

10. 现在，您必须将 Amazon S3 事件配置为 Lambda 触发器。

    在 Lambda 函数的配置选项卡/页面上，转到“触发器”，单击“添加触发器”。

11. 从可用触发器列表中选择 Amazon S3。
12. 要配置触发器，请选择相应的存储桶以从中触发函数。

    选择事件类型，即 PUT。或者，如果您只想处理具有特定名称或类型的文件，则可以指定前缀或后缀。

13. 单击“添加”激活该触发器并保存配置。

验证 Amazon SES 电子邮件地址

您必须先验证发件人和收件人的电子邮件地址，然后才能使用 Amazon SES 发送电子邮件。要实现此目的，应按照以下步骤进行：

1. 前往 Amazon SES 控制台。导航到“身份管理”，然后导航到“电子邮件地址”。
2. 单击“验证新的电子邮件地址”。输入要验证的电子邮件地址，然后单击“验证此电子邮件地址”。您将收到一封包含验证链接的电子邮件。单击该链接即可完成验证过程。
3. 在您验证了两个账户的电子邮件地址、设置了具有正确权限的 Lambda 函数并创建了一些示例客户数据之后，便可通过将测试图像上传到所选的 Amazon S3 存储桶来测试 Lambda 函数。

使用元数据将图像上传到 Amazon S3

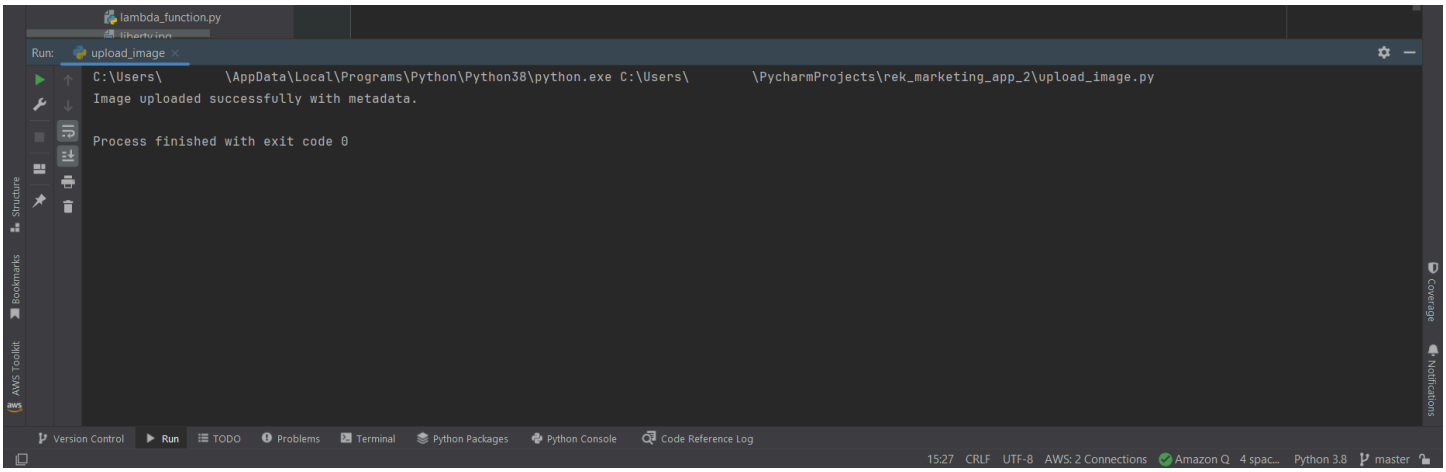
现在，我们可以使用 AWS 控制台或之前准备的脚本将测试图像上传到我们选择的 Amazon S3 存储桶。编写脚本，以将图像上传到之前在 Lambda 函数中指定的 Amazon S3 存储桶。在下面的代码示例中，将图像的路径指定为“upload_image_to_s3”的第一个参数，将存储桶名称指定为第二个参数，将上传图像的用户的账号指定为最后一个参数。

```
import boto3
def upload_image_to_s3(file_name, bucket, account_number):
    s3 = boto3.client('s3')
    try:
        with open(file_name, 'rb') as data:
            s3.upload_fileobj(
                Fileobj=data,
```

```
                Bucket=bucket,
                Key=file_name,
                ExtraArgs={
                    'Metadata': {'account_number': account_number}
                }
            )
        print("Image uploaded successfully with metadata.")
    except Exception as e:
        print("Failed to upload image")
        print(e)

# Usage
upload_image_to_s3('path-to-image-here', 's3-bucket-name-here', 'user-account-number-
here')
```

在此函数中，upload_fileobj 方法中的 ExtraArgs 参数用于在图像中包含用户定义的元数据 (account_number)。稍后可以使用此元数据 AWS 来相应地处理图像。

保存并运行该脚本。这将上传图像。



上传图像几分钟后，您应该会收到一封电子邮件，收件地址是您之前与此处指定的账户关联的地址。

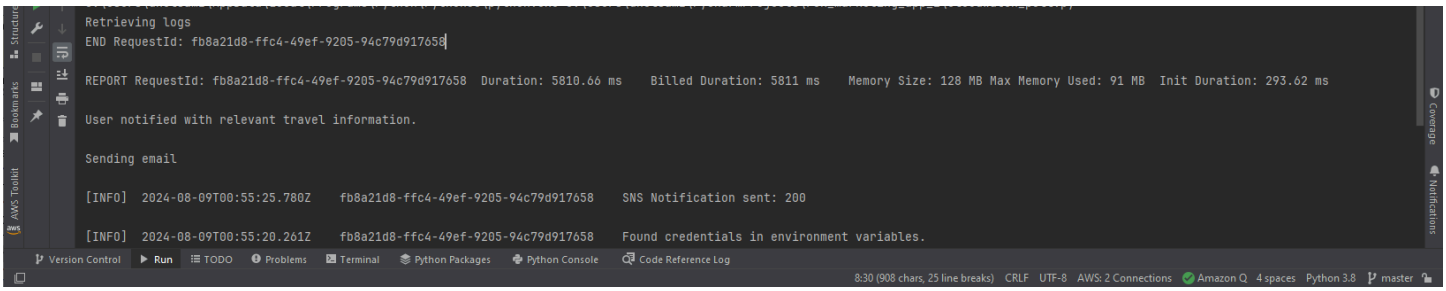
来自客户端的轮询 CloudWatch 日志

查看您的 Lambda 函数的 CloudWatch 日志，查看该函数是否按预期触发和执行。日志可以在 > 日志 > 日志组 CloudWatch > //lambda AWS/your_function_name 下找到。您也可以编写脚本来访问和打印日志。以下代码示例在 Lambda 组中轮询日志，打印出过去一小时内生成的日志。保存并运行代码。

```
import boto3
```

```
import time

def fetch_lambda_logs(log_group_name, start_time):
    client = boto3.client('logs')
    query = "fields @timestamp, @message | sort @timestamp desc | limit 20"
    start_query_response = client.start_query(
        logGroupName=log_group_name,
        startTime=int(start_time),
        endTime=int(time.time()),
        queryString=query,
    )
    query_id = start_query_response['queryId']
    response = None
    while response is None or response['status'] == 'Running':
        time.sleep(1)  # Wait for 1 second before checking the query status again
        response = client.get_query_results(queryId=query_id)
    return response['results']

# Usage
log_group = '/aws/lambda/RekMediaFunction'
logs = fetch_lambda_logs(log_group, int(time.time()) - 3600)  # Fetch logs from the
 last hour
print("Retrieving logs")
for log in logs:
    #print(log)
    print(log[1]['value'])
```

运行代码应该会打印出日志，并且您应该能够看到用户已通过包含相关旅行信息的电子邮件收到通知。



您已成功创建了一个应用程序，该应用程序能够检测上传到 Amazon S3 存储桶的图像中的标签，然后通过电子邮件向上传图像的用户发送促销信息。请务必删除任何不再需要的资源，这样就不会为它们付费。

使用 Lambda 和 Python 检测图像中的标签

AWS Lambda 是一项计算服务，无需预置或管理服务器即可使用它来运行代码。您可以从 Lambda 函数中调用 Rekognition API 操作。以下说明显示如何在 Python 中创建用来调用 `DetectLabels` 的 Lambda 函数。

Lambda 函数会调用 `DetectLabels` 并返回图像中检测到的一组标签和检测标签时所依据的置信度级别。

这些说明包括示例 Python 代码，该代码向您展示如何调用 Lambda 函数并向其提供来自 Amazon S3 存储桶或本地计算机的图像。

确保您选择的图像符合 Rekognition 的限制。有关图像文件类型和大小限制的信息，请参阅 Rekognition 中的指南和DetectLabels 配额以及 API 参考。

创建 Lambda 函数（控制台）

在此步骤中，您将创建一个空的 Lambda 函数和一个允许您的 Lambda 函数调用 `DetectLabels` 操作的 IAM 执行角色。在后续步骤中，您将添加源代码，也可以选择向 Lambda 函数添加层。

如果您使用的是存储在 Amazon S3 存储桶中的文档，则此步骤还演示了如何授予对存储您文档的存储桶的访问权限。

创建 AWS Lambda 函数（控制台）

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
2. 选择 Create function (创建函数)。有关更多信息，请参阅使用控制台创建 Lambda 函数。
3. 选择以下选项：
   - 选择从头开始创作。
   - 为函数名称输入一个值。
   - 对于运行时系统，请选择最新版本的 Python。
   - 对于架构，选择 x86_64。
4. 选择创建函数以创建 AWS Lambda 函数。
5. 在函数页面上，选择配置选项卡。
6. 在权限窗格的执行角色下，选择角色名称以在 IAM 控制台中打开该角色。
7. 在权限选项卡中，依次选择添加权限和创建内联策略。

8. 选择 JSON 选项卡，并将该策略替换为以下策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "rekognition:DetectLabels",
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DetectLabels"
        }
    ]
}
```

9. 选择查看策略。
10. 输入策略的名称，例如 DetectLabels-access。
11. 选择创建策略。
12. 如果您要在 Amazon S3 存储桶中存储用于分析的文档，则必须添加 Amazon S3 访问策略。为此，请在 AWS Lambda 控制台中重复步骤 7 到 11 并执行以下更改。

    a. 对于步骤 8，请使用以下策略。*bucket/folder path*替换为 Amazon S3 存储桶和您要分析的文档的文件夹路径。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

    b. 在步骤 10 中，选择不同的策略名称，例如 S3Bucket-access。

（可选）创建层（控制台）

您无需执行此步骤即可使用 Lambda 函数和调用 `DetectLabels`。

该DetectLabels操作作为适用于 Python 的 AWS SDK (Boto3) 的一部分包含在默认 Lambda Python 环境中。

如果您的 Lambda 函数的其他部分需要最新的 AWS 服务更新，而这些更新不在默认 Lambda Python 环境中，则可以执行此步骤将最新版本的 Boto3 SDK 作为层添加到您的函数中。

要将 SDK 添加为层，请先创建一个包含 Boto3 SDK 的 zip 文件存档。然后，创建一个层并将 zip 文件存档添加到该层。有关更多信息，请参阅组合使用层与 Lambda 函数。

创建并添加层（控制台）

1. 打开命令提示符并输入以下命令，使用最新版本的 AWS SDK 创建部署包。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. 记下您在本过程的步骤 8 中使用的压缩文件 (boto3-layer.zip) 的名称。
3. 打开 AWS Lambda 控制台，网址为https://console.aws.amazon.com/lambda/。
4. 在导航窗格中，选择层。
5. 选择创建层。
6. 为名称和描述输入值。
7. 对于代码输入种类，选择上传 .zip 文件并选择上传。
8. 在对话框中，选择您在本过程步骤 1 中创建的 zip 文件存档 (boto3-layer.zip)。
9. 对于兼容运行时系统，请选择最新版本的 Python。
10. 选择创建以创建层。
11. 选择导航窗格菜单图标。
12. 在导航窗格中，选择函数。
13. 在资源列表中，选择您之前在???中创建的函数。
14. 选择节点选项卡。
15. 在层部分，选择添加层。
16. 选择自定义层。
17. 在自定义层中，选择您在步骤 6 中输入的层名称。
18. 在版本中，选择层版本，该版本应为 1。
19. 选择 添加。

添加 Python 代码（控制台）

在此步骤中，您将使用 Lambda 控制台代码编辑器，向您的Lambda 函数添加 Python 代码。该代码使用 DetectLabels 操作检测图像中的标签。它返回在图像中检测到的一组标签，以及检测标签所依据的置信度级别。

您为 DetectLabels 操作提供的文档可以位于 Amazon S3 存储桶或本地计算机中。

添加 Python 代码（控制台）

1. 导航至代码选项卡。
2. 在代码编辑器中，将 lambda_function.py 中的代码替换为以下代码：

```
import boto3
import logging
from botocore.exceptions import ClientError
import json
import base64

# Instantiate logger
logger = logging.getLogger(__name__)

# connect to the Rekognition client
rekognition = boto3.client('rekognition')

def lambda_handler(event, context):

    try:
        image = None
        if 'S3Bucket' in event and 'S3Object' in event:
            s3 = boto3.resource('s3')
            s3_object = s3.Object(event['S3Bucket'], event['S3Object'])
            image = s3_object.get()['Body'].read()

        elif 'image' in event:
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image = img_b64decoded


        elif image is None:
            raise ValueError('Missing image, check image or bucket path.')
```

```
        else:
            raise ValueError("Only base 64 encoded image bytes or S3Object are
 supported.")

        response = rekognition.detect_labels(Image={'Bytes': image})
        lambda_response = {
            "statusCode": 200,
            "body": json.dumps(response)
        }
        labels = [label['Name'] for label in response['Labels']]
        print("Labels found:")
        print(labels)

    except ClientError as client_err:

       error_message = "Couldn't analyze image: " + client_err.response['Error']
['Message']

       lambda_response = {
           'statusCode': 400,
           'body': {
               "Error": client_err.response['Error']['Code'],
               "ErrorMessage": error_message
           }
       }
       logger.error("Error function %s: %s",
                    context.invoked_function_arn, error_message)


    except ValueError as val_error:

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error)
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response
```

3. 选择部署以部署您的 Lambda 函数。

添加 Python 代码（控制台）

现在您已创建 Lambda 函数，可以调用它来检测图像中的标签。

在此步骤中，您在计算机上运行 Python 代码，该代码会将本地图像或 Amazon S3 存储桶中的图像传递给您的 Lambda 函数。

确保在创建 Lambda 函数的同一 AWS 区域运行代码。您可以在 Lambda 控制台的函数详情页面的导航栏中查看 Lambda 函数的 AWS 区域。

如果 Lambda 函数返回超时错误，请延长 Lambda 函数的超时时间。有关更多信息，请参阅配置函数超时（控制台）。

有关从您的代码调用 Lambda 函数的更多信息，请参阅调用 AWS Lambda 函数。

试用您的 Lambda 函数

1. 执行以下操作（如果尚未这样做）：

   a. 确保用户拥有 `lambda:InvokeFunction` 权限。您可以使用以下策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InvokeLambda",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

   您可以从 Lambda 控制台中的函数概述，获取 Lambda 函数的 ARN。

   要提供访问权限，请为您的用户、组或角色添加权限：

   - 中的用户和群组 AWS IAM Identity Center：

创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

b. 安装和配置 AWS 适用于 Python 的开发工具包。有关更多信息，请参阅 步骤 2：设置 AWS CLI 和 AWS SDKs。

2. 将以下代码保存到名为 `client.py` 的文件中：

```
import boto3
import json
import base64
import pprint

# Replace with the name of your S3 bucket and image object key
bucket_name = "name of bucket"
object_key = "name of file in s3 bucket"
# If using a local file, supply the file name as the value of image_path below
image_path = ""

# Create session and establish connection to client['
session = boto3.Session(profile_name='developer-role')
s3 = session.client('s3', region_name="us-east-1")
lambda_client = session.client('lambda',  region_name="us-east-1")

# Replace with the name of your Lambda function
function_name = 'RekDetectLabels'

def analyze_image_local(img_path):

    print("Analyzing local image:")

    with open(img_path, 'rb') as image_file:
```

```
        image_bytes = image_file.read()
        data = base64.b64encode(image_bytes).decode("utf8")

        lambda_payload = {"image": data}

        # Invoke the Lambda function with the event payload
        response = lambda_client.invoke(
            FunctionName=function_name,
            Payload=(json.dumps(lambda_payload))
        )

        decoded = json.loads(response['Payload'].read().decode())
        pprint.pprint(decoded)

def analyze_image_s3(bucket_name, object_key):

    print("Analyzing image in S3 bucket:")

    # Load the image data from S3 into memory
    response = s3.get_object(Bucket=bucket_name, Key=object_key)
    image_data = response['Body'].read()
    image_data = base64.b64encode(image_data).decode("utf8")

     # Create the Lambda event payload
    event = {
        'S3Bucket': bucket_name,
        'S3Object': object_key,
        'ImageBytes': image_data
    }

    # Invoke the Lambda function with the event payload
    response = lambda_client.invoke(
            FunctionName=function_name,
            InvocationType='RequestResponse',
            Payload=json.dumps(event),
            )

    decoded = json.loads(response['Payload'].read().decode())
    pprint.pprint(decoded)

def main(path_to_image, name_s3_bucket, obj_key):

    if str(path_to_image) != "":
        analyze_image_local(path_to_image)
```

```
    else:
        analyze_image_s3(name_s3_bucket, obj_key)

if __name__ == "__main__":
    main(image_path, bucket_name, object_key)
```

3. 运行该代码。如果文档在 Amazon S3 存储桶中，请确保它与您之前在 ??? 步骤 12 中指定的存储桶相同。

   如果成功，代码会针对文档中检测到的每个块类型返回部分 JSON 响应。

使用 Amazon Rekognition 的代码示例 AWS SDKs

以下代码示例展示了如何将 Amazon Rekognition 与软件开发套件 (SDK) AWS 一起使用。本章中的代码示例旨在补充本指南其余部分中的代码示例。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

场景是向您展示如何通过在一个服务中调用多个函数或与其他 AWS 服务服务结合来完成特定任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

Hello Amazon Rekognition

以下代码示例展示了如何开始使用 Amazon Rekognition。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CMakeLists.txt CMake 文件的代码。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rekognition)

# Set this project's name.
project("hello_rekognition")
```

```
# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rekognition.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_rekognition.cpp 源文件的代码。

```
#include <aws/core/Aws.h>
#include <aws/rekognition/RekognitionClient.h>
#include <aws/rekognition/model/ListCollectionsRequest.h>
```

```
#include <iostream>

/*
 *  A "Hello Rekognition" starter application which initializes an Amazon
 Rekognition client and
 *  lists the Amazon Rekognition collections in the current account and region.
 *
 *  main function
 *
 *  Usage: 'hello_rekognition'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Rekognition::RekognitionClient rekognitionClient(clientConfig);
        Aws::Rekognition::Model::ListCollectionsRequest request;
        Aws::Rekognition::Model::ListCollectionsOutcome outcome =
                rekognitionClient.ListCollections(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String>& collectionsIds =
 outcome.GetResult().GetCollectionIds();
            if (!collectionsIds.empty()) {
                std::cout << "collectionsIds: " << std::endl;
                for (auto &collectionId : collectionsIds) {
                    std::cout << "- " << collectionId << std::endl;
                }
            } else {
                std::cout << "No collections found" << std::endl;
            }
        } else {
            std::cerr << "Error with ListCollections: " << outcome.GetError()
                      << std::endl;
        }
    }
```

```
    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考ListCollections中的。

代码示例

- 使用 Amazon Rekognition 的基本示例 AWS SDKs
  - Hello Amazon Rekognition
  - 使用 Amazon Rekognition 执行的操作 AWS SDKs
    - CompareFaces与 AWS SDK 或 CLI 配合使用
    - CreateCollection与 AWS SDK 或 CLI 配合使用
    - DeleteCollection与 AWS SDK 或 CLI 配合使用
    - DeleteFaces与 AWS SDK 或 CLI 配合使用
    - DescribeCollection与 AWS SDK 或 CLI 配合使用
    - DetectFaces与 AWS SDK 或 CLI 配合使用
    - DetectLabels与 AWS SDK 或 CLI 配合使用
    - DetectModerationLabels与 AWS SDK 或 CLI 配合使用
    - DetectText与 AWS SDK 或 CLI 配合使用
    - DisassociateFaces与 AWS SDK 或 CLI 配合使用
    - GetCelebrityInfo与 AWS SDK 或 CLI 配合使用
    - IndexFaces与 AWS SDK 或 CLI 配合使用
    - ListCollections与 AWS SDK 或 CLI 配合使用
    - ListFaces与 AWS SDK 或 CLI 配合使用
    - RecognizeCelebrities与 AWS SDK 或 CLI 配合使用
    - SearchFaces与 AWS SDK 或 CLI 配合使用
    - SearchFacesByImage与 AWS SDK 或 CLI 配合使用
- 亚马逊 Rekognition 使用场景 AWS SDKs
  - 构建 Amazon Rekognition 收藏并使用软件开发工具包在其中寻找面孔 AWS
  - 创建照片资产管理应用程序，让用户能够使用标签管理照片

- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的个人防护装备 AWS
- 使用软件开发工具包使用 Amazon Rekognition 检测和显示图像中的元素 AWS
- 使用 AWS SDK 检测图像中的人脸
- 使用亚马逊 Rekognition 和软件开发工具包检测视频中的信息 AWS
- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的物体 AWS
- 使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS
- 使用 SDK 保存 EXIF 和其他图像信息 AWS

使用 Amazon Rekognition 的基本示例 AWS SDKs

以下代码示例展示了如何使用 Amazon Rekognition 的基础知识。 AWS SDKs

示例

- Hello Amazon Rekognition
- 使用 Amazon Rekognition 执行的操作 AWS SDKs
  - CompareFaces与 AWS SDK 或 CLI 配合使用
  - CreateCollection与 AWS SDK 或 CLI 配合使用
  - DeleteCollection与 AWS SDK 或 CLI 配合使用
  - DeleteFaces与 AWS SDK 或 CLI 配合使用
  - DescribeCollection与 AWS SDK 或 CLI 配合使用
  - DetectFaces与 AWS SDK 或 CLI 配合使用
  - DetectLabels与 AWS SDK 或 CLI 配合使用
  - DetectModerationLabels与 AWS SDK 或 CLI 配合使用
  - DetectText与 AWS SDK 或 CLI 配合使用
  - DisassociateFaces与 AWS SDK 或 CLI 配合使用
  - GetCelebrityInfo与 AWS SDK 或 CLI 配合使用
  - IndexFaces与 AWS SDK 或 CLI 配合使用
  - ListCollections与 AWS SDK 或 CLI 配合使用
  - ListFaces与 AWS SDK 或 CLI 配合使用
  - RecognizeCelebrities与 AWS SDK 或 CLI 配合使用
  - SearchFaces与 AWS SDK 或 CLI 配合使用

- SearchFacesByImage与 AWS SDK 或 CLI 配合使用

Hello Amazon Rekognition

以下代码示例展示了如何开始使用 Amazon Rekognition。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

CMakeLists.txt CMake 文件的代码。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rekognition)

# Set this project's name.
project("hello_rekognition")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()
```

```
# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rekognition.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_rekognition.cpp 源文件的代码。

```
#include <aws/core/Aws.h>
#include <aws/rekognition/RekognitionClient.h>
#include <aws/rekognition/model/ListCollectionsRequest.h>
#include <iostream>

/*
 *  A "Hello Rekognition" starter application which initializes an Amazon
 Rekognition client and
 *  lists the Amazon Rekognition collections in the current account and region.
 *
 *  main function
 *
 *  Usage: 'hello_rekognition'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
```

```
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Rekognition::RekognitionClient rekognitionClient(clientConfig);
        Aws::Rekognition::Model::ListCollectionsRequest request;
        Aws::Rekognition::Model::ListCollectionsOutcome outcome =
                rekognitionClient.ListCollections(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String>& collectionsIds =
 outcome.GetResult().GetCollectionIds();
            if (!collectionsIds.empty()) {
                std::cout << "collectionsIds: " << std::endl;
                for (auto &collectionId : collectionsIds) {
                    std::cout << "- " << collectionId << std::endl;
                }
            } else {
                std::cout << "No collections found" << std::endl;
            }
        } else {
            std::cerr << "Error with ListCollections: " << outcome.GetError()
                      << std::endl;
        }
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考ListCollections中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon Rekognition 执行的操作 AWS SDKs

以下代码示例演示了如何使用执行单个 Amazon Rekognition 操作。 AWS SDKs每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

这些代码节选调用了 Amazon Rekognition API，是必须在上下文中运行的较大型程序的代码节选。您可以在亚马逊 Rekognition 使用场景 AWS SDKs 中结合上下文查看操作。

以下示例仅包括最常用的操作。有关完整列表，请参阅 Amazon Rekognition API 参考。

示例

- CompareFaces与 AWS SDK 或 CLI 配合使用
- CreateCollection与 AWS SDK 或 CLI 配合使用
- DeleteCollection与 AWS SDK 或 CLI 配合使用
- DeleteFaces与 AWS SDK 或 CLI 配合使用
- DescribeCollection与 AWS SDK 或 CLI 配合使用
- DetectFaces与 AWS SDK 或 CLI 配合使用
- DetectLabels与 AWS SDK 或 CLI 配合使用
- DetectModerationLabels与 AWS SDK 或 CLI 配合使用
- DetectText与 AWS SDK 或 CLI 配合使用
- DisassociateFaces与 AWS SDK 或 CLI 配合使用
- GetCelebrityInfo与 AWS SDK 或 CLI 配合使用
- IndexFaces与 AWS SDK 或 CLI 配合使用
- ListCollections与 AWS SDK 或 CLI 配合使用
- ListFaces与 AWS SDK 或 CLI 配合使用
- RecognizeCelebrities与 AWS SDK 或 CLI 配合使用
- SearchFaces与 AWS SDK 或 CLI 配合使用
- SearchFacesByImage与 AWS SDK 或 CLI 配合使用

`CompareFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CompareFaces`。

有关更多信息，请参阅比较图像中的人脸。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to compare faces in two images.
    /// </summary>
    public class CompareFaces
    {
        public static async Task Main()
        {
            float similarityThreshold = 70F;
            string sourceImage = "source.jpg";
            string targetImage = "target.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            Amazon.Rekognition.Model.Image imageSource = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(sourceImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
```

```
                Console.WriteLine($"Failed to load source image: {sourceImage}");
                return;
            }

            Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();

            try
            {
                using FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read);
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Failed to load target image: {targetImage}");
                Console.WriteLine(ex.Message);
                return;
            }

            var compareFacesRequest = new CompareFacesRequest
            {
                SourceImage = imageSource,
                TargetImage = imageTarget,
                SimilarityThreshold = similarityThreshold,
            };

            // Call operation
            var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

            // Display results
            compareFacesResponse.FaceMatches.ForEach(match =>
            {
                ComparedFace face = match.Face;
                BoundingBox position = face.BoundingBox;
                Console.WriteLine($"Face at {position.Left} {position.Top}
matches with {match.Similarity}% confidence.");
            });
```

```
            Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
 face(s) that did not match.");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CompareFaces中的。

CLI

AWS CLI

比较两张图像中的人脸

以下 compare-faces 命令将比较存储在 Amazon S3 存储桶中的两张图像中的人脸。

```
aws rekognition compare-faces \
    --source-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"source.jpg"}}' \
    --target-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"target.jpg"}}'
```

输出：

```
{
    "UnmatchedFaces": [],
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.12368916720151901,
                    "Top": 0.16007372736930847,
                    "Left": 0.5901257991790771,
                    "Height": 0.25140416622161865
                },
                "Confidence": 100.0,
                "Pose": {
                    "Yaw": -3.7351467609405518,
                    "Roll": -0.10309021919965744,
                    "Pitch": 0.8637830018997192
                },
                "Quality": {
```

```
                    "Sharpness": 95.51618957519531,
                    "Brightness": 65.29893493652344
                },
                "Landmarks": [
                    {
                        "Y": 0.26721030473709106,
                        "X": 0.6204193830490112,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26831310987472534,
                        "X": 0.6776827573776245,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3514654338359833,
                        "X": 0.6241428852081299,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.35258132219314575,
                        "X": 0.6713621020317078,
                        "Type": "mouthRight"
                    },
                    {
                        "Y": 0.3140771687030792,
                        "X": 0.6428444981575012,
                        "Type": "nose"
                    }
                ]
            },
            "Similarity": 100.0
        }
    ],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.12368916720151901,
            "Top": 0.16007372736930847,
            "Left": 0.5901257991790771,
            "Height": 0.25140416622161865
        },
        "Confidence": 100.0
    }
```

```
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的比较图像中的人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CompareFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import software.amazon.awssdk.core.SdkBytes;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CompareFaces {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceKey> <targetKey>

            Where:
```

```
                bucketName - The name of the S3 bucket where the images are
stored.
                sourceKey  - The S3 key (file name) for the source image.
                targetKey  - The S3 key (file name) for the target image.
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceKey = args[1];
        String targetKey = args[2];

        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
        compareTwoFaces(rekClient, bucketName, sourceKey, targetKey);
    }

   /**
    * Compares two faces from images stored in an Amazon S3 bucket using AWS
Rekognition.
    *
    * <p>This method takes two image keys from an S3 bucket and compares the
faces within them.
    * It prints out the confidence level of matched faces and reports the number
of unmatched faces.</p>
    *
    * @param rekClient   The {@link RekognitionClient} used to call AWS
Rekognition.
    * @param bucketName  The name of the S3 bucket containing the images.
    * @param sourceKey   The object key (file path) for the source image in the
S3 bucket.
    * @param targetKey   The object key (file path) for the target image in the
S3 bucket.
    * @throws RuntimeException If the Rekognition service returns an error.
    */
   public static void compareTwoFaces(RekognitionClient rekClient, String
bucketName, String sourceKey, String targetKey) {
        try {
            Float similarityThreshold = 70F;
```

```
            S3Object s3ObjectSource = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceKey)
                    .build();

            Image sourceImage = Image.builder()
                    .s3Object(s3ObjectSource)
                    .build();

            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(targetKey)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            CompareFacesRequest facesRequest = CompareFacesRequest.builder()
                    .sourceImage(sourceImage)
                    .targetImage(targetImage)
                    .similarityThreshold(similarityThreshold)
                    .build();

            // Compare the two images.
            CompareFacesResponse compareFacesResult =
rekClient.compareFaces(facesRequest);
            List<CompareFacesMatch> faceDetails =
compareFacesResult.faceMatches();

            for (CompareFacesMatch match : faceDetails) {
                ComparedFace face = match.face();
                BoundingBox position = face.boundingBox();
                System.out.println("Face at " + position.left().toString()
                        + " " + position.top()
                        + " matches with " + face.confidence().toString()
                        + "% confidence.");
            }

            List<ComparedFace> unmatchedFaces =
compareFacesResult.unmatchedFaces();
            System.out.println("There were " + unmatchedFaces.size() + " face(s)
that did not match.");
```

```
        } catch (RekognitionException e) {
            System.err.println("Error comparing faces: " +
 e.awsErrorDetails().errorMessage());
            throw new RuntimeException(e);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CompareFaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun compareTwoFaces(
    similarityThresholdVal: Float,
    sourceImageVal: String,
    targetImageVal: String,
) {
    val sourceBytes = (File(sourceImageVal).readBytes())
    val targetBytes = (File(targetImageVal).readBytes())

    // Create an Image object for the source image.
    val souImage =
        Image {
            bytes = sourceBytes
        }

    val tarImage =
        Image {
            bytes = targetBytes
        }

    val facesRequest =
        CompareFacesRequest {
```

```
            sourceImage = souImage
            targetImage = tarImage
            similarityThreshold = similarityThresholdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->

        val compareFacesResult = rekClient.compareFaces(facesRequest)
        val faceDetails = compareFacesResult.faceMatches

        if (faceDetails != null) {
            for (match: CompareFacesMatch in faceDetails) {
                val face = match.face
                val position = face?.boundingBox
                if (position != null) {
                    println("Face at ${position.left} ${position.top} matches
 with ${face.confidence} % confidence.")
                }
            }
        }

        val uncompared = compareFacesResult.unmatchedFaces
        if (uncompared != null) {
            println("There was ${uncompared.size} face(s) that did not match")
        }

        println("Source image rotation:
 ${compareFacesResult.sourceImageOrientationCorrection}")
        println("target image rotation:
 ${compareFacesResult.targetImageOrientationCorrection}")
    }
}
```

- 有关 API 的详细信息，请参阅适用CompareFaces于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value
 greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that
 have
                 a similarity value below the specified threshold.
```

```
        """
        try:
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity,
            )
            matches = [
                RekognitionFace(match["Face"]) for match in
response["FaceMatches"]
            ]
            unmatches = [RekognitionFace(face) for face in
response["UnmatchedFaces"]]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches),
                len(unmatches),
            )
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.",
                self.image_name,
                target_image.image_name,
            )
            raise
        else:
            return matches, unmatches
```

- 有关 API 的详细信息，请参阅适用CompareFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

CreateCollection与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateCollection`。

有关更多信息，请参阅创建集合。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses Amazon Rekognition to create a collection to which you can add
    /// faces using the IndexFaces operation.
    /// </summary>
    public class CreateCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Creating collection: " + collectionId);

            var createCollectionRequest = new CreateCollectionRequest
            {
                CollectionId = collectionId,
            };

            CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
            Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
            Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考CreateCollection中的。

CLI

AWS CLI

创建集合

以下 create-collection 命令创建具有指定名称的集合。

```
aws rekognition create-collection \
    --collection-id "MyCollection"
```

输出：

```
{
    "CollectionArn": "aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0",
    "StatusCode": 200
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的创建集合。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateCollection中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateCollection {
    public static void main(String[] args) {
        final String usage = """

            Usage: <collectionName>\s

            Where:
                collectionName - The name of the collection.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Creating collection: " + collectionId);
        createMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Creates a new Amazon Rekognition collection.
     *
```

```
     * @param rekClient    the Amazon Rekognition client used to interact with
 the Rekognition service
     * @param collectionId the unique identifier for the collection to be created
     */
    public static void createMyCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            CreateCollectionRequest collectionRequest =
 CreateCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

            CreateCollectionResponse collectionResponse =
 rekClient.createCollection(collectionRequest);
            System.out.println("CollectionArn: " +
 collectionResponse.collectionArn());
            System.out.println("Status code: " +
 collectionResponse.statusCode().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateCollection中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun createMyCollection(collectionIdVal: String) {
    val request =
        CreateCollectionRequest {
```

```
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.createCollection(request)
        println("Collection ARN is ${response.collectionArn}")
        println("Status code is ${response.statusCode}")
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateCollection于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def create_collection(self, collection_id):
        """
        Creates an empty collection.
```

```
        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id
            )
            response["CollectionId"] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection
```

- 有关 API 的详细信息，请参阅适用CreateCollection于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DeleteCollection与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteCollection。

有关更多信息，请参阅删除集合。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete an existing collection.
    /// </summary>
    public class DeleteCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Deleting collection: " + collectionId);

            var deleteCollectionRequest = new DeleteCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var deleteCollectionResponse = await
rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
            Console.WriteLine($"{collectionId}:
{deleteCollectionResponse.StatusCode}");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteCollection中的。

CLI

AWS CLI

删除集合

以下 `delete-collection` 命令将删除指定的集合。

```
aws rekognition delete-collection \
```

```
    --collection-id MyCollection
```

输出：

```
{
    "StatusCode": 200
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的删除集合。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteCollection中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId>\s
```

```
            Where:
                collectionId - The id of the collection to delete.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Deletes an Amazon Rekognition collection.
     *
     * @param rekClient      An instance of the {@link RekognitionClient} class,
which is used to interact with the Amazon Rekognition service.
     * @param collectionId   The ID of the collection to be deleted.
     */
    public static void deleteMyCollection(RekognitionClient rekClient, String
collectionId) {
        try {
            DeleteCollectionRequest deleteCollectionRequest =
DeleteCollectionRequest.builder()
                    .collectionId(collectionId)
                    .build();

            DeleteCollectionResponse deleteCollectionResponse =
rekClient.deleteCollection(deleteCollectionRequest);
            System.out.println(collectionId + ": " +
deleteCollectionResponse.statusCode().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
```

```
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteCollection中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteMyCollection(collectionIdVal: String) {
    val request =
        DeleteCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.deleteCollection(request)
        println("The collectionId status is ${response.statusCode}")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteCollection于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
```

```
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.",
self.collection_id)
            raise
```

- 有关 API 的详细信息，请参阅适用DeleteCollection于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteFaces`。

有关更多信息，请参阅从集中删除人脸。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete one or more faces from
    /// a Rekognition collection.
    /// </summary>
    public class DeleteFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" };

            var rekognitionClient = new AmazonRekognitionClient();

            var deleteFacesRequest = new DeleteFacesRequest()
            {
                CollectionId = collectionId,
                FaceIds = faces,
            };

            DeleteFacesResponse deleteFacesResponse = await
 rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
            deleteFacesResponse.DeletedFaces.ForEach(face =>
            {
                Console.WriteLine($"FaceID: {face}");
            });
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DeleteFaces中的。

CLI

AWS CLI

从集合中删除人脸

以下 delete-faces 命令将从集合中删除指定的人脸。

```
aws rekognition delete-faces \
    --collection-id MyCollection
    --face-ids '["0040279c-0178-436e-b70a-e61b074e96b0"]'
```

输出：

```
{
    "DeletedFaces": [
        "0040279c-0178-436e-b70a-e61b074e96b0"
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的从集合中删除人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId> <faceId>\s

            Where:
                collectionId - The id of the collection from which faces are
 deleted.\s
                faceId - The id of the face to delete.\s
           """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    /**
     * Deletes a face from the specified Amazon Rekognition collection.
     *
     * @param rekClient     an instance of the Amazon Rekognition client
     * @param collectionId  the ID of the collection from which the face should
 be deleted
     * @param faceId        the ID of the face to be deleted
     * @throws RekognitionException if an error occurs while deleting the face
     */
    public static void deleteFacesCollection(RekognitionClient rekClient,
```

```
            String collectionId,
            String faceId) {

        try {
            DeleteFacesRequest deleteFacesRequest = DeleteFacesRequest.builder()
                    .collectionId(collectionId)
                    .faceIds(faceId)
                    .build();

            rekClient.deleteFaces(deleteFacesRequest);
            System.out.println("The face was deleted from the collection.");

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteFaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteFacesCollection(
    collectionIdVal: String?,
    faceIdVal: String,
) {
    val deleteFacesRequest =
        DeleteFacesRequest {
            collectionId = collectionIdVal
            faceIds = listOf(faceIdVal)
        }
```

```
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        rekClient.deleteFaces(deleteFacesRequest)
        println("$faceIdVal was deleted from the collection")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteFaces于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
```

```
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def delete_faces(self, face_ids):
        """
        Deletes faces from the collection.

        :param face_ids: The list of IDs of faces to delete.
        :return: The list of IDs of faces that were deleted.
        """
        try:
            response = self.rekognition_client.delete_faces(
                CollectionId=self.collection_id, FaceIds=face_ids
            )
            deleted_ids = response["DeletedFaces"]
            logger.info(
                "Deleted %s faces from %s.", len(deleted_ids), self.collection_id
            )
        except ClientError:
            logger.exception("Couldn't delete faces from %s.",
self.collection_id)
            raise
        else:
            return deleted_ids
```

- 有关 API 的详细信息，请参阅适用DeleteFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DescribeCollection`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DescribeCollection`。

有关更多信息，请参阅描述集合。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
/// collection.
/// </summary>
public class DescribeCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        string collectionId = "MyCollection";
        Console.WriteLine($"Describing collection: {collectionId}");

        var describeCollectionRequest = new DescribeCollectionRequest()
        {
            CollectionId = collectionId,
        };

        var describeCollectionResponse = await
rekognitionClient.DescribeCollectionAsync(describeCollectionRequest);
        Console.WriteLine($"Collection ARN:
{describeCollectionResponse.CollectionARN}");
```

```
            Console.WriteLine($"Face count:
 {describeCollectionResponse.FaceCount}");
            Console.WriteLine($"Face model version:
 {describeCollectionResponse.FaceModelVersion}");
            Console.WriteLine($"Created:
 {describeCollectionResponse.CreationTimestamp}");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DescribeCollection中的。

CLI

AWS CLI

描述集合

以下 describe-collection 示例显示有关指定集合的详细信息。

```
aws rekognition describe-collection \
    --collection-id MyCollection
```

输出：

```
{
    "FaceCount": 200,
    "CreationTimestamp": 1569444828.274,
    "CollectionARN": "arn:aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0"
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的描述集合。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeCollection中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage:    <collectionName>

            Where:
                collectionName - The name of the Amazon Rekognition collection.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionName = args[0];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    /**
     * Describes an Amazon Rekognition collection.
     *
     * @param rekClient         The Amazon Rekognition client used to make the
 request.
     * @param collectionName    The name of the collection to describe.
     *
     * @throws RekognitionException If an error occurs while describing the
 collection.
     */
    public static void describeColl(RekognitionClient rekClient, String
 collectionName) {
        try {
            DescribeCollectionRequest describeCollectionRequest =
 DescribeCollectionRequest.builder()
                    .collectionId(collectionName)
                    .build();

            DescribeCollectionResponse describeCollectionResponse = rekClient
                    .describeCollection(describeCollectionRequest);
            System.out.println("Collection Arn : " +
 describeCollectionResponse.collectionARN());
            System.out.println("Created : " +
 describeCollectionResponse.creationTimestamp().toString());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DescribeCollection中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun describeColl(collectionName: String) {
    val request =
        DescribeCollectionRequest {
            collectionId = collectionName
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.describeCollection(request)
        println("The collection Arn is ${response.collectionArn}")
        println("The collection contains this many faces ${response.faceCount}")
    }
}
```

- 有关 API 的详细信息，请参阅适用DescribeCollection于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
```

```
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id
            )
            # Work around capitalization of Arn vs. ARN
```

```
            response["CollectionArn"] = response.get("CollectionARN")
            (
                self.collection_arn,
                self.face_count,
                self.created,
            ) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
self.collection_id)
            raise
        else:
            return self.to_dict()
```

- 有关 API 的详细信息，请参阅适用DescribeCollection于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectFaces`。

有关更多信息，请参阅检测图像中的人脸。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect faces within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
    /// </summary>
    public class DetectFaces
    {
        public static async Task Main()
        {
            string photo = "input.jpg";
            string bucket = "amzn-s3-demo-bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectFacesRequest = new DetectFacesRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },

                // Attributes can be "ALL" or "DEFAULT".
                // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and
 Quality.
                // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/
items/Rekognition/TFaceDetail.html
                Attributes = new List<string>() { "ALL" },
            };

            try
            {
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
```

```
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
left={face.BoundingBox.Top} width={face.BoundingBox.Width}
height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
{face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
{face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

显示图像中所有人脸的边界框信息。

```
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
```

```
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition
\target.jpg"; // "photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read);
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            int height;
            int width;

            // Used to extract original photo width/height
            using (var imageBitmap = new Bitmap(photo))
            {
                height = imageBitmap.Height;
                width = imageBitmap.Width;
            }

            Console.WriteLine("Image Information:");
            Console.WriteLine(photo);
            Console.WriteLine("Image Height: " + height);
            Console.WriteLine("Image Width: " + width);

            try
            {
                var detectFacesRequest = new DetectFacesRequest()
                {
                    Image = image,
                    Attributes = new List<string>() { "ALL" },
                };
```

```
                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"The detected face is estimated to be
 between {face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the
 image.</param>
        /// <param name="rotation">The rotation of the face's bounding box.</
param>
        public static void ShowBoundingBoxPositions(int imageHeight, int
 imageWidth, BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No estimated orientation. Check Exif data.");
                return;
```

```
            }

            // Calculate face position based on image orientation.
            switch (rotation)
            {
                case "ROTATE_0":
                    left = imageWidth * box.Left;
                    top = imageHeight * box.Top;
                    break;
                case "ROTATE_90":
                    left = imageHeight * (1 - (box.Top + box.Height));
                    top = imageWidth * box.Left;
                    break;
                case "ROTATE_180":
                    left = imageWidth - (imageWidth * (box.Left + box.Width));
                    top = imageHeight * (1 - (box.Top + box.Height));
                    break;
                case "ROTATE_270":
                    left = imageHeight * box.Top;
                    top = imageWidth * (1 - box.Left - box.Width);
                    break;
                default:
                    Console.WriteLine("No estimated orientation information.
Check Exif data.");
                    return;
            }

            // Display face location information.
            Console.WriteLine($"Left: {left}");
            Console.WriteLine($"Top: {top}");
            Console.WriteLine($"Face Width: {imageWidth * box.Width}");
            Console.WriteLine($"Face Height: {imageHeight * box.Height}");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DetectFaces中的。

CLI

AWS CLI

检测图像中的人脸

以下 `detect-faces` 命令将检测存储在 Amazon S3 存储桶中的指定图像中的人脸。

```
aws rekognition detect-faces \
    --image '{"S3Object":{"Bucket":"MyImageS3Bucket","Name":"MyFriend.jpg"}}' \
    --attributes "ALL"
```

输出：

```
{
    "FaceDetails": [
        {
            "Confidence": 100.0,
            "Eyeglasses": {
                "Confidence": 98.91107940673828,
                "Value": false
            },
            "Sunglasses": {
                "Confidence": 99.7966537475586,
                "Value": false
            },
            "Gender": {
                "Confidence": 99.56611633300781,
                "Value": "Male"
            },
            "Landmarks": [
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3514654338359833,
```

```
                    "X": 0.6241428852081299,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.35258132219314575,
                    "X": 0.6713621020317078,
                    "Type": "mouthRight"
                },
                {
                    "Y": 0.3140771687030792,
                    "X": 0.6428444981575012,
                    "Type": "nose"
                },
                {
                    "Y": 0.24662546813488007,
                    "X": 0.6001564860343933,
                    "Type": "leftEyeBrowLeft"
                },
                {
                    "Y": 0.24326619505882263,
                    "X": 0.6303644776344299,
                    "Type": "leftEyeBrowRight"
                },
                {
                    "Y": 0.23818562924861908,
                    "X": 0.6146903038024902,
                    "Type": "leftEyeBrowUp"
                },
                {
                    "Y": 0.24373626708984375,
                    "X": 0.6640064716339111,
                    "Type": "rightEyeBrowLeft"
                },
                {
                    "Y": 0.24877218902111053,
                    "X": 0.7025929093360901,
                    "Type": "rightEyeBrowRight"
                },
                {
                    "Y": 0.23938551545143127,
                    "X": 0.6823262572288513,
                    "Type": "rightEyeBrowUp"
                },
                {
```

```
                    "Y": 0.265746533870697,
                    "X": 0.6112898588180542,
                    "Type": "leftEyeLeft"
                },
                {
                    "Y": 0.2676128149032593,
                    "X": 0.6317071914672852,
                    "Type": "leftEyeRight"
                },
                {
                    "Y": 0.262735515832901,
                    "X": 0.6201658248901367,
                    "Type": "leftEyeUp"
                },
                {
                    "Y": 0.27025148272514343,
                    "X": 0.6206279993057251,
                    "Type": "leftEyeDown"
                },
                {
                    "Y": 0.268223375082016,
                    "X": 0.6658390760421753,
                    "Type": "rightEyeLeft"
                },
                {
                    "Y": 0.2672517001628876,
                    "X": 0.687832236289978,
                    "Type": "rightEyeRight"
                },
                {
                    "Y": 0.26383838057518005,
                    "X": 0.6769183874130249,
                    "Type": "rightEyeUp"
                },
                {
                    "Y": 0.27138751745224,
                    "X": 0.676596462726593,
                    "Type": "rightEyeDown"
                },
                {
                    "Y": 0.32283174991607666,
                    "X": 0.6350004076957703,
                    "Type": "noseLeft"
                },
```

```
                {
                    "Y": 0.3219289481639862,
                    "X": 0.6567046642303467,
                    "Type": "noseRight"
                },
                {
                    "Y": 0.3420318365097046,
                    "X": 0.6450609564781189,
                    "Type": "mouthUp"
                },
                {
                    "Y": 0.3664324879646301,
                    "X": 0.6455618143081665,
                    "Type": "mouthDown"
                },
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "leftPupil"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "rightPupil"
                },
                {
                    "Y": 0.26343393325805664,
                    "X": 0.5946047306060791,
                    "Type": "upperJawlineLeft"
                },
                {
                    "Y": 0.3543180525302887,
                    "X": 0.6044883728027344,
                    "Type": "midJawlineLeft"
                },
                {
                    "Y": 0.4084877669811249,
                    "X": 0.6477024555206299,
                    "Type": "chinBottom"
                },
                {
                    "Y": 0.3562754988670349,
                    "X": 0.707981526851654,
                    "Type": "midJawlineRight"
```

```
                },
                {
                    "Y": 0.26580461859703064,
                    "X": 0.7234612107276917,
                    "Type": "upperJawlineRight"
                }
            ],
            "Pose": {
                "Yaw": -3.7351467609405518,
                "Roll": -0.10309021919965744,
                "Pitch": 0.8637830018997192
            },
            "Emotions": [
                {
                    "Confidence": 8.74203109741211,
                    "Type": "SURPRISED"
                },
                {
                    "Confidence": 2.501944065093994,
                    "Type": "ANGRY"
                },
                {
                    "Confidence": 0.7378743290901184,
                    "Type": "DISGUSTED"
                },
                {
                    "Confidence": 3.5296201705932617,
                    "Type": "HAPPY"
                },
                {
                    "Confidence": 1.7162904739379883,
                    "Type": "SAD"
                },
                {
                    "Confidence": 9.518536567687988,
                    "Type": "CONFUSED"
                },
                {
                    "Confidence": 0.45474427938461304,
                    "Type": "FEAR"
                },
                {
                    "Confidence": 72.79895782470703,
                    "Type": "CALM"
```

```
                }
            ],
            "AgeRange": {
                "High": 48,
                "Low": 32
            },
            "EyesOpen": {
                "Confidence": 98.93987274169922,
                "Value": true
            },
            "BoundingBox": {
                "Width": 0.12368916720151901,
                "Top": 0.16007372736930847,
                "Left": 0.5901257991790771,
                "Height": 0.25140416622161865
            },
            "Smile": {
                "Confidence": 93.4493179321289,
                "Value": false
            },
            "MouthOpen": {
                "Confidence": 90.53053283691406,
                "Value": false
            },
            "Quality": {
                "Sharpness": 95.51618957519531,
                "Brightness": 65.29893493652344
            },
            "Mustache": {
                "Confidence": 89.85221099853516,
                "Value": false
            },
            "Beard": {
                "Confidence": 86.1991195678711,
                "Value": true
            }
        }
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的检测图像中的人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectFaces {
    public static void main(String[] args) {
        final String usage = """

            Usage:   <bucketName> <sourceImage>

            Where:
                bucketName = The name of the Amazon S3 bucket where the source
 image is stored.
                sourceImage - The name of the source image file in the Amazon S3
 bucket. (for example, pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
```

```
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectFacesinImage(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects faces in an image stored in an Amazon S3 bucket using the Amazon
Rekognition service.
     *
     * @param rekClient    The Amazon Rekognition client used to interact with
the Rekognition service.
     * @param bucketName   The name of the Amazon S3 bucket where the source
image is stored.
     * @param sourceImage  The name of the source image file in the Amazon S3
bucket.
     */
    public static void detectFacesinImage(RekognitionClient rekClient, String
bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                .bucket(bucketName)
                .name(sourceImage)
                .build();

            Image targetImage = Image.builder()
                .s3Object(s3ObjectTarget)
                .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                .attributes(Attribute.ALL)
                .image(targetImage)
                .build();

            DetectFacesResponse facesResponse =
rekClient.detectFaces(facesRequest);
            List<FaceDetail> faceDetails = facesResponse.faceDetails();
```

```
            for (FaceDetail face : faceDetails) {
                AgeRange ageRange = face.ageRange();
                System.out.println("The detected face is estimated to be between
 "
                        + ageRange.low().toString() + " and " +
 ageRange.high().toString()
                        + " years old.");

                System.out.println("There is a smile : " +
 face.smile().value().toString());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectFaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun detectFacesinImage(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectFacesRequest {
            attributes = listOf(Attribute.All)
            image = souImage
```

```
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectFaces(request)
        response.faceDetails?.forEach { face ->
            val ageRange = face.ageRange
            println("The detected face is estimated to be between
 ${ageRange?.low} and ${ageRange?.high} years old.")
            println("There is a smile ${face.smile?.value}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DetectFaces于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
```

```
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=["ALL"]
            )
            faces = [RekognitionFace(face) for face in response["FaceDetails"]]
            logger.info("Detected %s faces.", len(faces))
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces
```

- 有关 API 的详细信息，请参阅适用DetectFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DetectLabels与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DetectLabels。

有关更多信息，请参阅检测图像中的标签。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect labels within an image
/// stored in an Amazon Simple Storage Service (Amazon S3) bucket.
/// </summary>
public class DetectLabels
{
    public static async Task Main()
    {
        string photo = "del_river_02092020_01.jpg"; // "input.jpg";
        string bucket = "amzn-s3-demo-bucket"; // "bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectlabelsRequest = new DetectLabelsRequest
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket,
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F,
        };
```

```
            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

检测存储在计算机上的图像文件中的标签。

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
```

```
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
            catch (Exception)
            {
                Console.WriteLine("Failed to load file " + photo);
                return;
            }

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = image,
                MaxLabels = 10,
                MinConfidence = 77F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[DetectLabels](#)中的。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Detect instances of real-world entities within an image by using Amazon
 Rekognition
/*!
  \param imageBucket: The Amazon Simple Storage Service (Amazon S3) bucket
 containing an image.
  \param imageKey: The Amazon S3 key of an image object.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::Rekognition::detectLabels(const Aws::String &imageBucket,
                                       const Aws::String &imageKey,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::Rekognition::RekognitionClient rekognitionClient(clientConfiguration);

    Aws::Rekognition::Model::DetectLabelsRequest request;
    Aws::Rekognition::Model::S3Object s3Object;
    s3Object.SetBucket(imageBucket);
    s3Object.SetName(imageKey);

    Aws::Rekognition::Model::Image image;
    image.SetS3Object(s3Object);

    request.SetImage(image);

    const Aws::Rekognition::Model::DetectLabelsOutcome outcome =
 rekognitionClient.DetectLabels(request);

    if (outcome.IsSuccess()) {
        const Aws::Vector<Aws::Rekognition::Model::Label> &labels =
 outcome.GetResult().GetLabels();
        if (labels.empty()) {
```

```
            std::cout << "No labels detected" << std::endl;
        } else {
            for (const Aws::Rekognition::Model::Label &label: labels) {
                std::cout << label.GetName() << ": " << label.GetConfidence() <<
 std::endl;
            }
        }
    } else {
        std::cerr << "Error while detecting labels: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考DetectLabels中的。

CLI

AWS CLI

检测图像中的标签

以下 `detect-labels` 示例将检测存储在 Amazon S3 存储桶中的图像中的场景和对象。

```
aws rekognition detect-labels \
    --image '{"S3Object":{"Bucket":"bucket","Name":"image"}}'
```

输出：

```
{
    "Labels": [
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
```

```
                    "Name": "Transportation"
                }
            ],
            "Name": "Automobile"
        },
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Vehicle"
        },
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [],
            "Name": "Transportation"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Top": 0.5039216876029968,
                        "Left": 0.0037978808395564556,
                        "Height": 0.18528179824352264
                    },
                    "Confidence": 99.15271759033203
                },
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Top": 0.5251884460449219,
                        "Left": 0.7309805154800415,
                        "Height": 0.21577216684818268
                    },
                    "Confidence": 99.1286392211914
                },
                {
                    "BoundingBox": {
                        "Width": 0.14233611524105072,
```

```
                "Top": 0.5333095788955688,
                "Left": 0.6494812965393066,
                "Height": 0.15528248250484467
            },
            "Confidence": 98.48368072509766
        },
        {
            "BoundingBox": {
                "Width": 0.11086395382881165,
                "Top": 0.5354844927787781,
                "Left": 0.10355594009160995,
                "Height": 0.10271988064050674
            },
            "Confidence": 96.45606231689453
        },
        {
            "BoundingBox": {
                "Width": 0.06254628300666809,
                "Top": 0.5573825240135193,
                "Left": 0.46083059906959534,
                "Height": 0.053911514580249786
            },
            "Confidence": 93.65448760986328
        },
        {
            "BoundingBox": {
                "Width": 0.10105438530445099,
                "Top": 0.534368634223938,
                "Left": 0.5743985772132874,
                "Height": 0.12226245552301407
            },
            "Confidence": 93.06217193603516
        },
        {
            "BoundingBox": {
                "Width": 0.056389667093753815,
                "Top": 0.5235804319381714,
                "Left": 0.9427769780158997,
                "Height": 0.17163699865341187
            },
            "Confidence": 92.6864013671875
        },
        {
            "BoundingBox": {
```

```
                    "Width": 0.06003860384225845,
                    "Top": 0.5441341400146484,
                    "Left": 0.22409997880458832,
                    "Height": 0.06737709045410156
                },
                "Confidence": 90.4227066040039
            },
            {
                "BoundingBox": {
                    "Width": 0.02848697081208229,
                    "Top": 0.5107086896896362,
                    "Left": 0,
                    "Height": 0.19150497019290924
                },
                "Confidence": 86.65286254882812
            },
            {
                "BoundingBox": {
                    "Width": 0.04067881405353546,
                    "Top": 0.5566273927688599,
                    "Left": 0.316415935754776,
                    "Height": 0.03428703173995018
                },
                "Confidence": 85.36471557617188
            },
            {
                "BoundingBox": {
                    "Width": 0.043411049991846085,
                    "Top": 0.5394920110702515,
                    "Left": 0.18293385207653046,
                    "Height": 0.0893595889210701
                },
                "Confidence": 82.21705627441406
            },
            {
                "BoundingBox": {
                    "Width": 0.031183116137981415,
                    "Top": 0.5579366683959961,
                    "Left": 0.2853088080883026,
                    "Height": 0.03989990055561066
                },
                "Confidence": 81.0157470703125
            },
            {
```

```
                    "BoundingBox": {
                        "Width": 0.031113790348172188,
                        "Top": 0.5504819750785828,
                        "Left": 0.2580395042896271,
                        "Height": 0.056484755128622055
                    },
                    "Confidence": 56.13441467285156
                },
                {
                    "BoundingBox": {
                        "Width": 0.08586374670267105,
                        "Top": 0.5438792705535889,
                        "Left": 0.5128012895584106,
                        "Height": 0.08550430089235306
                    },
                    "Confidence": 52.37760925292969
                }
            ],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Car"
        },
        {
            "Instances": [],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Human"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.19360728561878204,
                        "Top": 0.35072067379951477,
                        "Left": 0.43734854459762573,
                        "Height": 0.2742200493812561
                    },
```

```
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Top": 0.5010883808135986,
                        "Left": 0.9155802130699158,
                        "Height": 0.06597328186035156
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Person"
        },
        {
            "Instances": [],
            "Confidence": 93.24951934814453,
            "Parents": [],
            "Name": "Machine"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.03561960905790329,
                        "Top": 0.6468243598937988,
                        "Left": 0.7850857377052307,
                        "Height": 0.08878646790981293
                    },
                    "Confidence": 93.24951934814453
                },
                {
                    "BoundingBox": {
                        "Width": 0.02217046171426773,
                        "Top": 0.6149078607559204,
                        "Left": 0.04757237061858177,
                        "Height": 0.07136218994855881
                    },
                    "Confidence": 91.5025863647461
                },
                {
                    "BoundingBox": {
```

```
                    "Width": 0.016197510063648224,
                    "Top": 0.6274210214614868,
                    "Left": 0.6472989320755005,
                    "Height": 0.04955997318029404
                },
                "Confidence": 85.14686584472656
            },
            {
                "BoundingBox": {
                    "Width": 0.020207518711686134,
                    "Top": 0.6348286867141724,
                    "Left": 0.7295016646385193,
                    "Height": 0.07059963047504425
                },
                "Confidence": 83.34547424316406
            },
            {
                "BoundingBox": {
                    "Width": 0.020280985161662102,
                    "Top": 0.6171894669532776,
                    "Left": 0.08744934946298599,
                    "Height": 0.05297485366463661
                },
                "Confidence": 79.9981460571289
            },
            {
                "BoundingBox": {
                    "Width": 0.018318990245461464,
                    "Top": 0.623889148235321,
                    "Left": 0.6836880445480347,
                    "Height": 0.06730121374130249
                },
                "Confidence": 78.87144470214844
            },
            {
                "BoundingBox": {
                    "Width": 0.021310249343514442,
                    "Top": 0.6167286038398743,
                    "Left": 0.004064912907779217,
                    "Height": 0.08317798376083374
                },
                "Confidence": 75.89361572265625
            },
            {
```

```
                    "BoundingBox": {
                        "Width": 0.03604431077837944,
                        "Top": 0.7030032277107239,
                        "Left": 0.9254803657531738,
                        "Height": 0.04569442570209503
                    },
                    "Confidence": 64.402587890625
                },
                {
                    "BoundingBox": {
                        "Width": 0.009834849275648594,
                        "Top": 0.5821820497512817,
                        "Left": 0.28094568848609924,
                        "Height": 0.01964157074689865
                    },
                    "Confidence": 62.79907989501953
                },
                {
                    "BoundingBox": {
                        "Width": 0.01475677452981472,
                        "Top": 0.6137543320655823,
                        "Left": 0.5950819253921509,
                        "Height": 0.039063986390829086
                    },
                    "Confidence": 59.40483474731445
                }
            ],
            "Confidence": 93.24951934814453,
            "Parents": [
                {
                    "Name": "Machine"
                }
            ],
            "Name": "Wheel"
        },
        {
            "Instances": [],
            "Confidence": 92.61514282226562,
            "Parents": [],
            "Name": "Road"
        },
        {
            "Instances": [],
            "Confidence": 92.37877655029297,
```

```
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Sport"
        },
        {
            "Instances": [],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Sports"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.12326609343290329,
                        "Top": 0.6332163214683533,
                        "Left": 0.44815489649772644,
                        "Height": 0.058117982000112534
                    },
                    "Confidence": 92.37877655029297
                }
            ],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                },
                {
                    "Name": "Sport"
                }
            ],
            "Name": "Skateboard"
        },
        {
            "Instances": [],
            "Confidence": 90.62931060791016,
            "Parents": [
```

```
                {
                    "Name": "Person"
                }
            ],
            "Name": "Pedestrian"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
            "Name": "Asphalt"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
            "Name": "Tarmac"
        },
        {
            "Instances": [],
            "Confidence": 88.23201751708984,
            "Parents": [],
            "Name": "Path"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Urban"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Town"
        },
        {
```

```
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Building"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "City"
        },
        {
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking Lot"
        },
        {
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
```

```
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking"
        },
        {
            "Instances": [],
            "Confidence": 74.37590026855469,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                },
                {
                    "Name": "City"
                }
            ],
            "Name": "Downtown"
        },
        {
            "Instances": [],
            "Confidence": 69.84622955322266,
            "Parents": [
                {
                    "Name": "Road"
                }
            ],
            "Name": "Intersection"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Sports Car"
                },
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
```

```
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Coupe"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Sports Car"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Sidewalk"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Pavement"
        },
        {
```

```
            "Instances": [],
            "Confidence": 55.58770751953125,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Neighborhood"
        }
    ],
    "LabelModelVersion": "2.0"
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的检测图像中的标签。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectLabels中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceImage>

            Where:
                bucketName - The name of the Amazon S3 bucket where the image is
 stored
                sourceImage - The name of the image file (for example, pic1.png).
\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0] ;
        String sourceImage = args[1] ;
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects the labels in an image stored in an Amazon S3 bucket using the
 Amazon Rekognition service.
     *
     * @param rekClient     the Amazon Rekognition client used to make the
 detection request
     * @param bucketName    the name of the Amazon S3 bucket where the image is
 stored
     * @param sourceImage   the name of the image file to be analyzed
```

```
     */
    public static void detectImageLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            DetectLabelsRequest detectLabelsRequest =
 DetectLabelsRequest.builder()
                    .image(souImage)
                    .maxLabels(10)
                    .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label : labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectLabels中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun detectImageLabels(sourceImage: String) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }
    val request =
        DetectLabelsRequest {
            image = souImage
            maxLabels = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectLabels(request)
        response.labels?.forEach { label ->
            println("${label.name} : ${label.confidence}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DetectLabels于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
            )
            labels = [RekognitionLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels
```

- 有关 API 的详细信息，请参阅适用DetectLabels于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectModerationLabels`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectModerationLabels`。

有关更多信息，请参阅检测不适宜的图像。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect unsafe content in a
/// JPEG or PNG format image.
/// </summary>
public class DetectModerationLabels
{
    public static async Task Main(string[] args)
    {
        string photo = "input.jpg";
        string bucket = "amzn-s3-demo-bucket";

        var rekognitionClient = new AmazonRekognitionClient();

        var detectModerationLabelsRequest = new
DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
```

```
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MinConfidence = 60F,
            };

            try
            {
                var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
                {
                    Console.WriteLine($"Label: {label.Name}");
                    Console.WriteLine($"Confidence: {label.Confidence}");
                    Console.WriteLine($"Parent: {label.ParentName}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[DetectModerationLabels](#)中的。

CLI

AWS CLI

检测图像中不安全的内容

以下 `detect-moderation-labels` 命令将检测存储在 Amazon S3 存储桶中的指定图像中不安全的内容。

```
aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=gun.jpg}"
```

输出：

```
{
    "ModerationModelVersion": "3.0",
    "ModerationLabels": [
        {
            "Confidence": 97.29618072509766,
            "ParentName": "Violence",
            "Name": "Weapon Violence"
        },
        {
            "Confidence": 97.29618072509766,
            "ParentName": "",
            "Name": "Violence"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的检测不安全的图像。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectModerationLabels中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectModerationLabels {

    public static void main(String[] args) {
        final String usage = """
            Usage:  <bucketName>  <sourceImage>

            Where:
                bucketName - The name of the S3 bucket where the images are
 stored.
                sourceImage - The name of the image (for example, pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectModLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects moderation labels in an image stored in an Amazon S3 bucket.
     *
```

```
    * @param rekClient      the Amazon Rekognition client to use for the
 detection
    * @param bucketName     the name of the Amazon S3 bucket where the image is
 stored
    * @param sourceImage    the name of the image file to be analyzed
    *
    * @throws RekognitionException if there is an error during the image
 detection process
    */
   public static void detectModLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
       try {
           S3Object s3ObjectTarget = S3Object.builder()
                   .bucket(bucketName)
                   .name(sourceImage)
                   .build();

           Image targetImage = Image.builder()
                   .s3Object(s3ObjectTarget)
                   .build();

           DetectModerationLabelsRequest moderationLabelsRequest =
 DetectModerationLabelsRequest.builder()
                   .image(targetImage)
                   .minConfidence(60F)
                   .build();

           DetectModerationLabelsResponse moderationLabelsResponse = rekClient
                   .detectModerationLabels(moderationLabelsRequest);
           List<ModerationLabel> labels =
 moderationLabelsResponse.moderationLabels();
           System.out.println("Detected labels for image");
           for (ModerationLabel label : labels) {
               System.out.println("Label: " + label.name()
                       + "\n Confidence: " + label.confidence().toString() + "%"
                       + "\n Parent:" + label.parentName());
           }

       } catch (RekognitionException e) {
           e.printStackTrace();
           System.exit(1);
       }
   }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectModerationLabels中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun detectModLabels(sourceImage: String) {
    val myImage =
        Image {
            this.bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectModerationLabelsRequest {
            image = myImage
            minConfidence = 60f
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectModerationLabels(request)
        response.moderationLabels?.forEach { label ->
            println("Label: ${label.name} - Confidence: ${label.confidence} %
 Parent: ${label.parentName}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DetectModerationLabels于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify
 content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image
            )
```

```
            labels = [
                RekognitionModerationLabel(label)
                for label in response["ModerationLabels"]
            ]
            logger.info(
                "Found %s moderation labels in %s.", len(labels), self.image_name
            )
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name
            )
            raise
        else:
            return labels
```

- 有关 API 的详细信息，请参阅适用DetectModerationLabels于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetectText`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetectText`。

有关更多信息，请参阅检测图像中的文本。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect text in an image. The
    /// example was created using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class DetectText
    {
        public static async Task Main()
        {
            string photo = "Dad_photographer.jpg"; // "input.jpg";
            string bucket = "amzn-s3-demo-bucket"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectTextRequest = new DetectTextRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
            };

            try
            {
                DetectTextResponse detectTextResponse = await
rekognitionClient.DetectTextAsync(detectTextRequest);
                Console.WriteLine($"Detected lines and words for {photo}");
                detectTextResponse.TextDetections.ForEach(text =>
                {
                    Console.WriteLine($"Detected: {text.DetectedText}");
                    Console.WriteLine($"Confidence: {text.Confidence}");
                    Console.WriteLine($"Id : {text.Id}");
                    Console.WriteLine($"Parent Id: {text.ParentId}");
                    Console.WriteLine($"Type: {text.Type}");
                });
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e.Message);
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考DetectText中的。

CLI

AWS CLI

检测图像中的文本

以下 `detect-text` 命令将检测指定图像中的文本。

```
aws rekognition detect-text \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePicture.jpg"}}'
```

输出：

```
{
    "TextDetections": [
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.24624845385551453,
                    "Top": 0.28288066387176514,
                    "Left": 0.391388863325119,
                    "Height": 0.022687450051307678
                },
                "Polygon": [
                    {
                        "Y": 0.28288066387176514,
                        "X": 0.391388863325119
                    },
                    {
                        "Y": 0.2826388478279114,
                        "X": 0.6376373171806335
                    },
```

```
                    {
                        "Y": 0.30532628297805786,
                        "X": 0.637677013874054
                    },
                    {
                        "Y": 0.305568128824234,
                        "X": 0.39142853021621704
                    }
                ]
            },
            "Confidence": 94.35709381103516,
            "DetectedText": "ESTD 1882",
            "Type": "LINE",
            "Id": 0
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933889865875244,
                    "Top": 0.32603850960731506,
                    "Left": 0.34534579515457153,
                    "Height": 0.07126858830451965
                },
                "Polygon": [
                    {
                        "Y": 0.32603850960731506,
                        "X": 0.34534579515457153
                    },
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.3976001739501953,
                        "X": 0.684575080871582
                    },
                    {
                        "Y": 0.3973070979118347,
                        "X": 0.345236212015152
                    }
                ]
            },
            "Confidence": 99.95779418945312,
            "DetectedText": "BRAINS",
```

```
            "Type": "LINE",
            "Id": 1
        },
        {
            "Confidence": 97.22098541259766,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.061079490929841995,
                    "Top": 0.2843210697174072,
                    "Left": 0.391391396522522,
                    "Height": 0.021029088646173477
                },
                "Polygon": [
                    {
                        "Y": 0.2843210697174072,
                        "X": 0.391391396522522
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.4524524509906769
                    },
                    {
                        "Y": 0.3038259446620941,
                        "X": 0.4534534513950348
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.3923923969268799
                    }
                ]
            },
            "DetectedText": "ESTD",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 2
        },
        {
            "Confidence": 91.49320983886719,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.07007007300853729,
                    "Top": 0.2828207015991211,
                    "Left": 0.5675675868988037,
                    "Height": 0.02250562608242035
```

```
                },
                "Polygon": [
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.5675675868988037
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.5675675868988037
                    }
                ]
            },
            "DetectedText": "1882",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 3
        },
        {
            "Confidence": 99.95779418945312,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933934569358826,
                    "Top": 0.32633158564567566,
                    "Left": 0.3453453481197357,
                    "Height": 0.07127484679222107
                },
                "Polygon": [
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.3453453481197357
                    },
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.684684693813324
                    },
                    {
```

```
                    "Y": 0.39759939908981323,
                    "X": 0.6836836934089661
                },
                {
                    "Y": 0.39684921503067017,
                    "X": 0.3453453481197357
                }
            ]
        },
        "DetectedText": "BRAINS",
        "ParentId": 1,
        "Type": "WORD",
        "Id": 4
    }
  ]
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DetectText中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectText {
    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:   <bucketName> <sourceImage>\n" +
            "\n" +
            "Where:\n" +
            "   bucketName - The name of the S3 bucket where the image is stored
\n" +
            "   sourceImage - The path to the image that contains text (for
 example, pic1.png). \n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectTextLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects text labels in an image stored in an S3 bucket using Amazon
 Rekognition.
     *
     * @param rekClient    an instance of the Amazon Rekognition client
     * @param bucketName   the name of the S3 bucket where the image is stored
     * @param sourceImage  the name of the image file in the S3 bucket
     * @throws RekognitionException if an error occurs while calling the Amazon
 Rekognition API
     */
```

```
    public static void detectTextLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            DetectTextRequest textRequest = DetectTextRequest.builder()
                    .image(souImage)
                    .build();

            DetectTextResponse textResponse = rekClient.detectText(textRequest);
            List<TextDetection> textCollection = textResponse.textDetections();
            System.out.println("Detected lines and words");
            for (TextDetection text : textCollection) {
                System.out.println("Detected: " + text.detectedText());
                System.out.println("Confidence: " +
 text.confidence().toString());
                System.out.println("Id : " + text.id());
                System.out.println("Parent Id: " + text.parentId());
                System.out.println("Type: " + text.type());
                System.out.println();
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DetectText中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun detectTextLabels(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectTextRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectText(request)
        response.textDetections?.forEach { text ->
            println("Detected: ${text.detectedText}")
            println("Confidence: ${text.confidence}")
            println("Id: ${text.id}")
            println("Parent Id:  ${text.parentId}")
            println("Type: ${text.type}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用DetectText于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in
 response["TextDetections"]]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
```

```
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

- 有关 API 的详细信息，请参阅适用DetectText于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DisassociateFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DisassociateFaces`。

CLI

AWS CLI

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
  --user-id user-id --collection-id collection-name --region region-name
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DisassociateFaces中的。

Python

适用于 Python 的 SDK（Boto3）

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user
```

```
    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

if __name__ == "__main__":
    main()
```

- 有关 API 的详细信息，请参阅适用DisassociateFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`GetCelebrityInfo`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `GetCelebrityInfo`。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to retrieve information about the
    /// celebrity identified by the supplied celebrity Id.
    /// </summary>
    public class CelebrityInfo
    {
        public static async Task Main()
        {
            string celebId = "nnnnnnnn";

            var rekognitionClient = new AmazonRekognitionClient();

            var celebrityInfoRequest = new GetCelebrityInfoRequest
            {
                Id = celebId,
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
```

```
                Console.WriteLine(url);
            });
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考GetCelebrityInfo中的。

CLI

AWS CLI

获取有关名人的信息

以下 `get-celebrity-info` 命令显示有关指定名人的信息：`id` 参数来自先前对 `recognize-celebrities` 的调用。

```
aws rekognition get-celebrity-info --id nnnnnnn
```

输出：

```
{
    "Name": "Celeb A",
    "Urls": [
        "www.imdb.com/name/aaaaaaaaa"
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的获取有关名人的信息。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考GetCelebrityInfo中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`IndexFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `IndexFaces`。

有关更多信息，请参阅将人脸添加到集合中。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "amzn-s3-demo-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var indexFacesRequest = new IndexFacesRequest
        {
```

```
                Image = image,
                CollectionId = collectionId,
                ExternalImageId = photo,
                DetectionAttributes = new List<string>() { "ALL" },
            };

            IndexFacesResponse indexFacesResponse = await
 rekognitionClient.IndexFacesAsync(indexFacesRequest);

            Console.WriteLine($"{photo} added");
            foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
            {
                Console.WriteLine($"Face detected: Faceid is
 {faceRecord.Face.FaceId}");
            }
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考IndexFaces中的。

CLI

AWS CLI

将人脸添加到集合

以下 `index-faces` 命令将在图像中找到的人脸添加到指定的集合中。

```
aws rekognition index-faces \
    --image '{"S3Object":{"Bucket":"MyVideoS3Bucket","Name":"MyPicture.jpg"}}' \
    --collection-id MyCollection \
    --max-faces 1 \
    --quality-filter "AUTO" \
    --detection-attributes "ALL" \
    --external-image-id "MyPicture.jpg"
```

输出：

```
{
    "FaceRecords": [
        {
```

```
            "FaceDetail": {
                "Confidence": 99.993408203125,
                "Eyeglasses": {
                    "Confidence": 99.11750030517578,
                    "Value": false
                },
                "Sunglasses": {
                    "Confidence": 99.98249053955078,
                    "Value": false
                },
                "Gender": {
                    "Confidence": 99.92769622802734,
                    "Value": "Male"
                },
                "Landmarks": [
                    {
                        "Y": 0.26750367879867554,
                        "X": 0.6202793717384338,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26642778515815735,
                        "X": 0.6787431836128235,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.31361380219459534,
                        "X": 0.6421601176261902,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.3495299220085144,
                        "X": 0.6216195225715637,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.35194727778434753,
                        "X": 0.669899046421051,
                        "Type": "mouthRight"
                    },
                    {
                        "Y": 0.26844894886016846,
                        "X": 0.6210268139839172,
                        "Type": "leftPupil"
```

```
                    },
                    {
                        "Y": 0.26707562804222107,
                        "X": 0.6817160844802856,
                        "Type": "rightPupil"
                    },
                    {
                        "Y": 0.24834522604942322,
                        "X": 0.6018546223640442,
                        "Type": "leftEyeBrowLeft"
                    },
                    {
                        "Y": 0.24397172033786774,
                        "X": 0.6172008514404297,
                        "Type": "leftEyeBrowUp"
                    },
                    {
                        "Y": 0.24677404761314392,
                        "X": 0.6339119076728821,
                        "Type": "leftEyeBrowRight"
                    },
                    {
                        "Y": 0.24582654237747192,
                        "X": 0.6619398593902588,
                        "Type": "rightEyeBrowLeft"
                    },
                    {
                        "Y": 0.23973053693771362,
                        "X": 0.6804757118225098,
                        "Type": "rightEyeBrowUp"
                    },
                    {
                        "Y": 0.24441994726657867,
                        "X": 0.6978968977928162,
                        "Type": "rightEyeBrowRight"
                    },
                    {
                        "Y": 0.2695908546447754,
                        "X": 0.6085202693939209,
                        "Type": "leftEyeLeft"
                    },
                    {
                        "Y": 0.26716896891593933,
                        "X": 0.6315826177597046,
```

```
                        "Type": "leftEyeRight"
                    },
                    {
                        "Y": 0.26289820671081543,
                        "X": 0.6202316880226135,
                        "Type": "leftEyeUp"
                    },
                    {
                        "Y": 0.27123287320137024,
                        "X": 0.6205548048019409,
                        "Type": "leftEyeDown"
                    },
                    {
                        "Y": 0.2668408751487732,
                        "X": 0.6663622260093689,
                        "Type": "rightEyeLeft"
                    },
                    {
                        "Y": 0.26741549372673035,
                        "X": 0.6910083889961243,
                        "Type": "rightEyeRight"
                    },
                    {
                        "Y": 0.2614026665687561,
                        "X": 0.6785826086997986,
                        "Type": "rightEyeUp"
                    },
                    {
                        "Y": 0.27075251936912537,
                        "X": 0.6789616942405701,
                        "Type": "rightEyeDown"
                    },
                    {
                        "Y": 0.3211299479007721,
                        "X": 0.6324167847633362,
                        "Type": "noseLeft"
                    },
                    {
                        "Y": 0.32276326417922974,
                        "X": 0.6558475494384766,
                        "Type": "noseRight"
                    },
                    {
                        "Y": 0.34385165572166443,
```

```
                        "X": 0.6444970965385437,
                        "Type": "mouthUp"
                    },
                    {
                        "Y": 0.3671635091304779,
                        "X": 0.6459195017814636,
                        "Type": "mouthDown"
                    }
                ],
                "Pose": {
                    "Yaw": -9.54541015625,
                    "Roll": -0.5709401965141296,
                    "Pitch": 0.6045494675636292
                },
                "Emotions": [
                    {
                        "Confidence": 39.90074157714844,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 23.38753890991211,
                        "Type": "CALM"
                    },
                    {
                        "Confidence": 5.840933322906494,
                        "Type": "CONFUSED"
                    }
                ],
                "AgeRange": {
                    "High": 63,
                    "Low": 45
                },
                "EyesOpen": {
                    "Confidence": 99.80887603759766,
                    "Value": true
                },
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "Smile": {
                    "Confidence": 99.69740295410156,
```

```
                    "Value": false
                },
                "MouthOpen": {
                    "Confidence": 99.97393798828125,
                    "Value": false
                },
                "Quality": {
                    "Sharpness": 95.54405975341797,
                    "Brightness": 63.867706298828125
                },
                "Mustache": {
                    "Confidence": 97.05007934570312,
                    "Value": false
                },
                "Beard": {
                    "Confidence": 87.34505462646484,
                    "Value": false
                }
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.993408203125,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            }
        }
    ],
    "UnindexedFaces": [],
    "FaceModelVersion": "3.0",
    "OrientationCorrection": "ROTATE_0"
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的将人脸添加到集合中。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考IndexFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AddFacesToCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId> <sourceImage> <bucketName>

            Where:
                collectionName - The name of the collection.
                sourceImage - The name of the image (for example, pic1.png).
                bucketName - The name of the S3 bucket.
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
```

```
        String bucketName = args[2];;
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        addToCollection(rekClient, collectionId, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Adds a face from an image to an Amazon Rekognition collection.
     *
     * @param rekClient     the Amazon Rekognition client
     * @param collectionId  the ID of the collection to add the face to
     * @param bucketName    the name of the Amazon S3 bucket containing the image
     * @param sourceImage   the name of the image file to add to the collection
     * @throws RekognitionException if there is an error while interacting with
the Amazon Rekognition service
     */
    public static void addToCollection(RekognitionClient rekClient, String
collectionId, String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            IndexFacesRequest facesRequest = IndexFacesRequest.builder()
                    .collectionId(collectionId)
                    .image(targetImage)
                    .maxFaces(1)
                    .qualityFilter(QualityFilter.AUTO)
                    .detectionAttributes(Attribute.DEFAULT)
                    .build();

            IndexFacesResponse facesResponse =
rekClient.indexFaces(facesRequest);
            System.out.println("Results for the image");
            System.out.println("\n Faces indexed:");
```

```
            List<FaceRecord> faceRecords = facesResponse.faceRecords();
            for (FaceRecord faceRecord : faceRecords) {
                System.out.println("  Face ID: " + faceRecord.face().faceId());
                System.out.println("  Location:" +
 faceRecord.faceDetail().boundingBox().toString());
            }

            List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
            System.out.println("Faces not indexed:");
            for (UnindexedFace unindexedFace : unindexedFaces) {
                System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
                System.out.println("  Reasons:");
                for (Reason reason : unindexedFace.reasons()) {
                    System.out.println("Reason:  " + reason);
                }
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考IndexFaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun addToCollection(
    collectionIdVal: String?,
    sourceImage: String,
) {
```

```
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        IndexFacesRequest {
            collectionId = collectionIdVal
            image = souImage
            maxFaces = 1
            qualityFilter = QualityFilter.Auto
            detectionAttributes = listOf(Attribute.Default)
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val facesResponse = rekClient.indexFaces(request)

        // Display the results.
        println("Results for the image")
        println("\n Faces indexed:")
        facesResponse.faceRecords?.forEach { faceRecord ->
            println("Face ID: ${faceRecord.face?.faceId}")
            println("Location: ${faceRecord.faceDetail?.boundingBox}")
        }

        println("Faces not indexed:")
        facesResponse.unindexedFaces?.forEach { unindexedFace ->
            println("Location: ${unindexedFace.faceDetail?.boundingBox}")
            println("Reasons:")

            unindexedFace.reasons?.forEach { reason ->
                println("Reason:  $reason")
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用IndexFaces于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
```

```
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id,
                Image=image.image,
                ExternalImageId=image.image_name,
                MaxFaces=max_faces,
                DetectionAttributes=["ALL"],
            )
            indexed_faces = [
                RekognitionFace({**face["Face"], **face["FaceDetail"]})
                for face in response["FaceRecords"]
            ]
            unindexed_faces = [
                RekognitionFace(face["FaceDetail"])
                for face in response["UnindexedFaces"]
            ]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
                len(indexed_faces),
                image.image_name,
                len(unindexed_faces),
            )
        except ClientError:
            logger.exception("Couldn't index faces in image %s.",
image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces
```

- 有关 API 的详细信息，请参阅适用IndexFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListCollections`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListCollections`。

有关更多信息，请参阅列出集合。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses Amazon Rekognition to list the collection IDs in the
/// current account.
/// </summary>
public class ListCollections
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        var listCollectionsRequest = new ListCollectionsRequest
```

```
            {
                MaxResults = limit,
            };

            var listCollectionsResponse = new ListCollectionsResponse();

            do
            {
                if (listCollectionsResponse is not null)
                {
                    listCollectionsRequest.NextToken =
listCollectionsResponse.NextToken;
                }

                listCollectionsResponse = await
rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

                listCollectionsResponse.CollectionIds.ForEach(id =>
                {
                    Console.WriteLine(id);
                });
            }
            while (listCollectionsResponse.NextToken is not null);
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListCollections中的。

CLI

AWS CLI

列出可用的集合

以下`list-collections`命令列出了 AWS 账户中的可用集合。

```
aws rekognition list-collections
```

输出：

```
{
```

```
    "FaceModelVersions": [
        "2.0",
        "3.0",
        "3.0",
        "3.0",
        "4.0",
        "1.0",
        "3.0",
        "4.0",
        "4.0",
        "4.0"
    ],
    "CollectionIds": [
        "MyCollection1",
        "MyCollection2",
        "MyCollection3",
        "MyCollection4",
        "MyCollection5",
        "MyCollection6",
        "MyCollection7",
        "MyCollection8",
        "MyCollection9",
        "MyCollection10"
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的列出集合。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListCollections中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListCollections {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Listing collections");
        listAllCollections(rekClient);
        rekClient.close();
    }

    public static void listAllCollections(RekognitionClient rekClient) {
        try {
            ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
                    .maxResults(10)
                    .build();

            ListCollectionsResponse response =
 rekClient.listCollections(listCollectionsRequest);
            List<String> collectionIds = response.collectionIds();
            for (String resultId : collectionIds) {
                System.out.println(resultId);
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
```

```
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListCollections中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun listAllCollections() {
    val request =
        ListCollectionsRequest {
            maxResults = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listCollections(request)
        response.collectionIds?.forEach { resultId ->
            println(resultId)
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListCollections于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client


    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({"CollectionId": col_id},
 self.rekognition_client)
                for col_id in response["CollectionIds"]
            ]
        except ClientError:
```

```
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections
```

- 有关 API 的详细信息，请参阅适用ListCollections于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListFaces`。

有关更多信息，请参阅列出集合中的人脸。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to retrieve the list of faces
/// stored in a collection.
/// </summary>
public class ListFaces
{
    public static async Task Main()
    {
```

```
            string collectionId = "MyCollection2";

            var rekognitionClient = new AmazonRekognitionClient();

            var listFacesResponse = new ListFacesResponse();
            Console.WriteLine($"Faces in collection {collectionId}");

            var listFacesRequest = new ListFacesRequest
            {
                CollectionId = collectionId,
                MaxResults = 1,
            };

            do
            {
                listFacesResponse = await
 rekognitionClient.ListFacesAsync(listFacesRequest);
                listFacesResponse.Faces.ForEach(face =>
                {
                    Console.WriteLine(face.FaceId);
                });

                listFacesRequest.NextToken = listFacesResponse.NextToken;
            }
            while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考ListFaces中的。

CLI

AWS CLI

列出集合中的人脸

以下 `list-faces` 命令将列出指定集合中的人脸。

```
aws rekognition list-faces \
    --collection-id MyCollection
```

输出：


```
{
    "FaceModelVersion": "3.0",
    "Faces": [
        {
            "BoundingBox": {
                "Width": 0.5216310024261475,
                "Top": 0.3256250023841858,
                "Left": 0.13394300639629364,
                "Height": 0.3918749988079071
            },
            "FaceId": "0040279c-0178-436e-b70a-e61b074e96b0",
            "ExternalImageId": "image1.jpg",
            "Confidence": 100.0,
            "ImageId": "f976e487-3719-5e2d-be8b-ea2724c26991"
        },
        {
            "BoundingBox": {
                "Width": 0.5074880123138428,
                "Top": 0.3774999976158142,
                "Left": 0.18302799761295319,
                "Height": 0.3812499940395355
            },
            "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
            "ExternalImageId": "image2.jpg",
            "Confidence": 99.99930572509766,
            "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
        },
        {
            "BoundingBox": {
                "Width": 0.5574039816856384,
                "Top": 0.37187498807907104,
                "Left": 0.14559100568294525,
                "Height": 0.4181250035762787
            },
            "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
            "ExternalImageId": "image3.jpg",
            "Confidence": 99.99960327148438,
            "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
        },
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
```

```
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
            "ExternalImageId": "image4.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
        },
        {
            "BoundingBox": {
                "Width": 0.5307819843292236,
                "Top": 0.2862499952316284,
                "Left": 0.1564060002565384,
                "Height": 0.3987500071525574
            },
            "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
            "ExternalImageId": "image5.jpg",
            "Confidence": 99.99970245361328,
            "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
        },
        {
            "BoundingBox": {
                "Width": 0.5773710012435913,
                "Top": 0.34437501430511475,
                "Left": 0.12396000325679779,
                "Height": 0.4337500035762787
            },
            "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
            "ExternalImageId": "image6.jpg",
            "Confidence": 100.0,
            "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
        },
        {
            "BoundingBox": {
                "Width": 0.5349419713020325,
                "Top": 0.29124999046325684,
                "Left": 0.16389399766921997,
                "Height": 0.40187498927116394
            },
            "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
            "ExternalImageId": "image7.jpg",
            "Confidence": 99.99979400634766,
            "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
```

```
        },
        {
            "BoundingBox": {
                "Width": 0.41499999165534973,
                "Top": 0.09187500178813934,
                "Left": 0.28083300590515137,
                "Height": 0.3112500011920929
            },
            "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
            "ExternalImageId": "image8.jpg",
            "Confidence": 99.99769592285156,
            "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
        },
        {
            "BoundingBox": {
                "Width": 0.48166701197624207,
                "Top": 0.20999999344348907,
                "Left": 0.21250000596046448,
                "Height": 0.36125001311302185
            },
            "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
            "ExternalImageId": "image9.jpg",
            "Confidence": 99.99949645996094,
            "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
        },
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
            "ExternalImageId": "image10.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的列出集合中的人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListFacesInCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId>

                Where:
                   collectionId - The name of the collection.\s
                """;

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId);
        rekClient.close();
    }

    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                    .collectionId(collectionId)
                    .maxResults(10)
                    .build();

            ListFacesResponse facesResponse = rekClient.listFaces(facesRequest);
            List<Face> faces = facesResponse.faces();
            for (Face face : faces) {
                System.out.println("Confidence level there is a face: " +
 face.confidence());
                System.out.println("The face Id value is " + face.faceId());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListFaces中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun listFacesCollection(collectionIdVal: String?) {
    val request =
        ListFacesRequest {
            collectionId = collectionIdVal
            maxResults = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listFaces(request)
        response.faces?.forEach { face ->
            println("Confidence level there is a face: ${face.confidence}")
            println("The face Id value is ${face.faceId}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListFaces于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
```

```
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
```

```
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results
            )
            faces = [RekognitionFace(face) for face in response["Faces"]]
            logger.info(
                "Found %s faces in collection %s.", len(faces),
self.collection_id
            )
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id
            )
            raise
        else:
            return faces
```

- 有关 API 的详细信息，请参阅适用ListFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`RecognizeCelebrities`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `RecognizeCelebrities`。

有关更多信息，请参阅识别图像中的名人。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.IO;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to identify celebrities in a photo.
    /// </summary>
    public class CelebritiesInImage
    {
        public static async Task Main(string[] args)
        {
            string photo = "moviestars.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

            var img = new Amazon.Rekognition.Model.Image();
            byte[] data = null;
            try
            {
                using var fs = new FileStream(photo, FileMode.Open,
FileAccess.Read);
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
            }
            catch (Exception)
            {
                Console.WriteLine($"Failed to load file {photo}");
                return;
            }

            img.Bytes = new MemoryStream(data);
            recognizeCelebritiesRequest.Image = img;

            Console.WriteLine($"Looking for celebrities in image {photo}\n");

            var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);


Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
            recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
            {
```

```
                Console.WriteLine($"Celebrity recognized: {celeb.Name}");
                Console.WriteLine($"Celebrity ID: {celeb.Id}");
                BoundingBox boundingBox = celeb.Face.BoundingBox;
                Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
                Console.WriteLine("Further information (if available):");
                celeb.Urls.ForEach(url =>
                {
                    Console.WriteLine(url);
                });
            });


Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count}
face(s) were unrecognized.");
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考RecognizeCelebrities中的。

CLI

AWS CLI

识别图像中的名人

以下 `recognize-celebrities` 命令将识别存储在 Amazon S3 存储桶中的指定图像中的名人。

```
aws rekognition recognize-celebrities \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=moviestars.jpg}"
```

输出：

```
{
    "UnrecognizedFaces": [
        {
            "BoundingBox": {
                "Width": 0.14416666328907013,
```

```
                "Top": 0.07777778059244156,
                "Left": 0.625,
                "Height": 0.2746031880378723
            },
            "Confidence": 99.9990234375,
            "Pose": {
                "Yaw": 10.80408763885498,
                "Roll": -12.761146545410156,
                "Pitch": 10.96889877319336
            },
            "Quality": {
                "Sharpness": 94.1185531616211,
                "Brightness": 79.18367004394531
            },
            "Landmarks": [
                {
                    "Y": 0.18220913410186768,
                    "X": 0.6702951788902283,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.16337193548679352,
                    "X": 0.7188183665275574,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.20739148557186127,
                    "X": 0.7055801749229431,
                    "Type": "nose"
                },
                {
                    "Y": 0.2889308035373688,
                    "X": 0.687512218952179,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.2706988751888275,
                    "X": 0.7250053286552429,
                    "Type": "mouthRight"
                }
            ]
        }
    ],
    "CelebrityFaces": [
```

```
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.14000000059604645,
                    "Top": 0.1190476194024086,
                    "Left": 0.82833331823349,
                    "Height": 0.2666666805744171
                },
                "Confidence": 99.99359130859375,
                "Pose": {
                    "Yaw": -10.509642601013184,
                    "Roll": -14.51749324798584,
                    "Pitch": 13.799399375915527
                },
                "Quality": {
                    "Sharpness": 78.74752044677734,
                    "Brightness": 42.201324462890625
                },
                "Landmarks": [
                    {
                        "Y": 0.2290833294391632,
                        "X": 0.8709492087364197,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.20639978349208832,
                        "X": 0.9153988361358643,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.25417643785476685,
                        "X": 0.8907724022865295,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.32729196548461914,
                        "X": 0.8876466155052185,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.3115464746952057,
                        "X": 0.9238573312759399,
                        "Type": "mouthRight"
```

```
                    }
                ]
            },
            "Name": "Celeb A",
            "Urls": [
                "www.imdb.com/name/aaaaaaaaa"
            ],
            "Id": "1111111"
        },
        {
            "MatchConfidence": 97.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.13333334028720856,
                    "Top": 0.24920634925365448,
                    "Left": 0.4449999928474426,
                    "Height": 0.2539682686328888
                },
                "Confidence": 99.99979400634766,
                "Pose": {
                    "Yaw": 6.557040691375732,
                    "Roll": -7.316643714904785,
                    "Pitch": 9.272967338562012
                },
                "Quality": {
                    "Sharpness": 83.23492431640625,
                    "Brightness": 78.83267974853516
                },
                "Landmarks": [
                    {
                        "Y": 0.3625510632991791,
                        "X": 0.48898839950561523,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.35366007685661316,
                        "X": 0.5313721299171448,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3894785940647125,
                        "X": 0.5173314809799194,
                        "Type": "nose"
                    },
```

```
                    {
                        "Y": 0.44889405369758606,
                        "X": 0.5020005702972412,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.4408611059188843,
                        "X": 0.5351271629333496,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb B",
            "Urls": [
                "www.imdb.com/name/bbbbbbbbb"
            ],
            "Id": "2222222"
        },
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.12416666746139526,
                    "Top": 0.2968254089355469,
                    "Left": 0.2150000035762787,
                    "Height": 0.23650793731212616
                },
                "Confidence": 99.99958801269531,
                "Pose": {
                    "Yaw": 7.801797866821289,
                    "Roll": -8.326810836791992,
                    "Pitch": 7.844768047332764
                },
                "Quality": {
                    "Sharpness": 86.93206024169922,
                    "Brightness": 79.81291198730469
                },
                "Landmarks": [
                    {
                        "Y": 0.4027804136276245,
                        "X": 0.2575301229953766,
                        "Type": "eyeLeft"
                    },
                    {
```

```
                    "Y": 0.3934555947780609,
                    "X": 0.2956969439983368,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.4309830069541931,
                    "X": 0.2837020754814148,
                    "Type": "nose"
                },
                {
                    "Y": 0.48186683654785156,
                    "X": 0.26812544465065,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.47338807582855225,
                    "X": 0.29905644059181213,
                    "Type": "mouthRight"
                }
            ]
        },
        "Name": "Celeb C",
        "Urls": [
            "www.imdb.com/name/ccccccccc"
        ],
        "Id": "3333333"
    },
    {
        "MatchConfidence": 97.0,
        "Face": {
            "BoundingBox": {
                "Width": 0.11916666477918625,
                "Top": 0.3698412775993347,
                "Left": 0.008333333767950535,
                "Height": 0.22698412835597992
            },
            "Confidence": 99.99999237060547,
            "Pose": {
                "Yaw": 16.38478660583496,
                "Roll": -1.0260354280471802,
                "Pitch": 5.975185394287109
            },
            "Quality": {
                "Sharpness": 83.23492431640625,
```

```
                    "Brightness": 61.408443450927734
                },
                "Landmarks": [
                    {
                        "Y": 0.4632347822189331,
                        "X": 0.049406956881284714,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.46388113498687744,
                        "X": 0.08722897619009018,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.5020678639411926,
                        "X": 0.0758260041475296,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.544157862663269,
                        "X": 0.054029736667871475,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.5463630557060242,
                        "X": 0.08464983850717545,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb D",
            "Urls": [
                "www.imdb.com/name/ddddddddd"
            ],
            "Id": "4444444"
        }
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的识别图像中的名人。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考RecognizeCelebrities中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

import software.amazon.awssdk.services.rekognition.model.*;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class RecognizeCelebrities {
    public static void main(String[] args) {
        final String usage = """
                Usage:   <bucketName> <sourceImage>

                Where:
                   bucketName - The name of the S3 bucket where the images are
 stored.
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 2) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];;
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Locating celebrities in " + sourceImage);
        recognizeAllCelebrities(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Recognizes all celebrities in an image stored in an Amazon S3 bucket.
     *
     * @param rekClient    the Amazon Rekognition client used to perform the
celebrity recognition operation
     * @param bucketName   the name of the Amazon S3 bucket where the source
image is stored
     * @param sourceImage  the name of the source image file stored in the Amazon
S3 bucket
     */
    public static void recognizeAllCelebrities(RekognitionClient rekClient,
String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                .bucket(bucketName)
                .name(sourceImage)
                .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            RecognizeCelebritiesRequest request =
RecognizeCelebritiesRequest.builder()
                    .image(souImage)
                    .build();
```

```
            RecognizeCelebritiesResponse result =
 rekClient.recognizeCelebrities(request);
            List<Celebrity> celebs = result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.
\n");
            for (Celebrity celebrity : celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
                System.out.println("Celebrity ID: " + celebrity.id());

                System.out.println("Further information (if available):");
                for (String url : celebrity.urls()) {
                    System.out.println(url);
                }
                System.out.println();
            }
            System.out.println(result.unrecognizedFaces().size() + " face(s) were
 unrecognized.");

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考RecognizeCelebrities中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun recognizeAllCelebrities(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
```

```
        }

    val request =
        RecognizeCelebritiesRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.recognizeCelebrities(request)
        response.celebrityFaces?.forEach { celebrity ->
            println("Celebrity recognized: ${celebrity.name}")
            println("Celebrity ID:${celebrity.id}")
            println("Further information (if available):")
            celebrity.urls?.forEach { url ->
                println(url)
            }
        }
        println("${response.unrecognizedFaces?.size} face(s) were unrecognized.")
    }
}
```

- 有关 API 的详细信息，请参阅适用RecognizeCelebrities于 K otlin 的AWS SDK API 参考。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
```

```
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    def recognize_celebrities(self):
        """
        Detects celebrities in the image.

        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response =
self.rekognition_client.recognize_celebrities(Image=self.image)
            celebrities = [
                RekognitionCelebrity(celeb) for celeb in
response["CelebrityFaces"]
            ]
            other_faces = [
                RekognitionFace(face) for face in response["UnrecognizedFaces"]
            ]
            logger.info(
                "Found %s celebrities and %s other faces in %s.",
                len(celebrities),
                len(other_faces),
                self.image_name,
            )
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.",
self.image_name)
            raise
        else:
            return celebrities, other_faces
```

- 有关 API 的详细信息，请参阅适用RecognizeCelebrities于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SearchFaces`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SearchFaces`。

有关更多信息，请参阅搜索人脸（面容 ID）。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to find faces in an image that
/// match the face Id provided in the method request.
/// </summary>
public class SearchFacesMatchingId
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

        var rekognitionClient = new AmazonRekognitionClient();
```

```
            // Search collection for faces matching the face id.
            var searchFacesRequest = new SearchFacesRequest
            {
                CollectionId = collectionId,
                FaceId = faceId,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesResponse searchFacesResponse = await
 rekognitionClient.SearchFacesAsync(searchFacesRequest);

            Console.WriteLine("Face matching faceId " + faceId);

            Console.WriteLine("Matche(s): ");
            searchFacesResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
 {face.Similarity}");
            });
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考SearchFaces中的。

CLI

AWS CLI

搜索集合中与人脸 ID 匹配的人脸

以下 `search-faces` 命令将搜索集合中与指定人脸 ID 相匹配的人脸。

```
aws rekognition search-faces \
    --face-id 8d3cfc70-4ba8-4b36-9644-90fba29c2dac \
    --collection-id MyCollection
```

输出：

```
{
```

```
    "SearchedFaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
    "FaceModelVersion": "3.0",
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.48166701197624207,
                    "Top": 0.20999999344348907,
                    "Left": 0.21250000596046448,
                    "Height": 0.36125001311302185
                },
                "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
                "ExternalImageId": "image1.jpg",
                "Confidence": 99.99949645996094,
                "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
            },
            "Similarity": 99.30997467041016
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            },
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
                "ExternalImageId": "image3.jpg",
                "Confidence": 99.99340057373047,
```

```
                "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
            },
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5349419713020325,
                    "Top": 0.29124999046325684,
                    "Left": 0.16389399766921997,
                    "Height": 0.40187498927116394
                },
                "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
                "ExternalImageId": "image9.jpg",
                "Confidence": 99.99979400634766,
                "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
            },
            "Similarity": 96.73158264160156
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5307819843292236,
                    "Top": 0.2862499952316284,
                    "Left": 0.1564060002565384,
                    "Height": 0.3987500071525574
                },
                "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
                "ExternalImageId": "image10.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
            },
            "Similarity": 96.48291015625
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5074880123138428,
                    "Top": 0.3774999976158142,
                    "Left": 0.18302799761295319,
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
```

```
                "Confidence": 99.99930572509766,
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
            },
            "Similarity": 96.43287658691406
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
                    "Top": 0.37187498807907104,
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 95.25305938720703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 95.22837829589844
        }
    ]
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的使用人脸 ID 搜索人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SearchFaces中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingImageCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
```

```
                    collectionId - The id of the collection. \s
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s

                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage);
        rekClient.close();
    }

    public static void searchFaceInCollection(RekognitionClient rekClient, String
 collectionId, String sourceImage) {
        try {
            InputStream sourceStream = new FileInputStream(new
 File(sourceImage));
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                    .bytes(sourceBytes)
                    .build();

            SearchFacesByImageRequest facesByImageRequest =
 SearchFacesByImageRequest.builder()
                    .image(souImage)
                    .maxFaces(10)
                    .faceMatchThreshold(70F)
                    .collectionId(collectionId)
                    .build();

            SearchFacesByImageResponse imageResponse =
 rekClient.searchFacesByImage(facesByImageRequest);
            System.out.println("Faces matching in the collection");
```

```
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face : faceImageMatches) {
                System.out.println("The similarity level is  " +
 face.similarity());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SearchFaces中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
```

```
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list
does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id,
                FaceId=face_id,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            faces = [RekognitionFace(face["Face"]) for face in
response["FaceMatches"]]
```

```
            logger.info(
                "Found %s faces in %s that match %s.",
                len(faces),
                self.collection_id,
                face_id,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                face_id,
            )
            raise
        else:
            return faces
```

- 有关 API 的详细信息，请参阅适用SearchFaces于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SearchFacesByImage`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SearchFacesByImage`。

有关更多信息，请参阅搜索人脸（图像）。

.NET

适用于 .NET 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
```

```
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to search for images matching those
    /// in a collection.
    /// </summary>
    public class SearchFacesMatchingImage
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string bucket = "amzn-s3-demo-bucket";
            string photo = "input.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            // Get an image object from S3 bucket.
            var image = new Image()
            {
                S3Object = new S3Object()
                {
                    Bucket = bucket,
                    Name = photo,
                },
            };

            var searchFacesByImageRequest = new SearchFacesByImageRequest()
            {
                CollectionId = collectionId,
                Image = image,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " +
photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
```

```
        }
    }
```

- 有关 API 的详细信息，请参阅 适用于 .NET 的 AWS SDK API 参考[SearchFacesByImage](#)中的。

CLI

AWS CLI

搜索集合中与图像中最大人脸匹配的人脸。

以下 search-faces-by-image 命令将搜索集合中与指定图像中最大人脸相匹配的人脸。

```
aws rekognition search-faces-by-image \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePerson.jpg"}}' \
    --collection-id MyFaceImageCollection

{
    "SearchedFaceBoundingBox": {
        "Width": 0.18562500178813934,
        "Top": 0.1618015021085739,
        "Left": 0.5575000047683716,
        "Height": 0.24770642817020416
    },
    "SearchedFaceConfidence": 99.993408203125,
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            },
```

```
            "Similarity": 99.97913360595703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
                "ExternalImageId": "image3.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
            },
            "Similarity": 99.97913360595703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.41499999165534973,
                    "Top": 0.09187500178813934,
                    "Left": 0.28083300590515137,
                    "Height": 0.3112500011920929
                },
                "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
                "ExternalImageId": "image2.jpg",
                "Confidence": 99.99769592285156,
                "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
            },
            "Similarity": 99.18069458007812
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.48166701197624207,
                    "Top": 0.20999999344348907,
                    "Left": 0.21250000596046448,
                    "Height": 0.36125001311302185
                },
                "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
                "ExternalImageId": "image1.jpg",
                "Confidence": 99.99949645996094,
                "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
```

```
            },
            "Similarity": 98.66607666015625
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5349419713020325,
                    "Top": 0.29124999046325684,
                    "Left": 0.16389399766921997,
                    "Height": 0.40187498927116394
                },
                "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
                "ExternalImageId": "image9.jpg",
                "Confidence": 99.99979400634766,
                "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
            },
            "Similarity": 98.24278259277344
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5307819843292236,
                    "Top": 0.2862499952316284,
                    "Left": 0.1564060002565384,
                    "Height": 0.3987500071525574
                },
                "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
                "ExternalImageId": "image10.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
            },
            "Similarity": 98.10665893554688
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5074880123138428,
                    "Top": 0.3774999976158142,
                    "Left": 0.18302799761295319,
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
                "Confidence": 99.99930572509766,
```

```
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
            },
            "Similarity": 98.10526275634766
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
                    "Top": 0.37187498807907104,
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 97.94659423828125
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 97.93476867675781
        }
    ],

 "FaceModelVersion": "3.0"
}
```

有关更多信息，请参阅《Amazon Rekognition 开发人员指南》中的使用图像搜索人脸。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SearchFacesByImage中的。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingIdCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
                   collectionId - The id of the collection. \s
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
```

```
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId);
        rekClient.close();
    }

    public static void searchFacebyId(RekognitionClient rekClient, String
 collectionId, String faceId) {
        try {
            SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
                    .collectionId(collectionId)
                    .faceId(faceId)
                    .faceMatchThreshold(70F)
                    .maxFaces(2)
                    .build();

            SearchFacesResponse imageResponse =
 rekClient.searchFaces(searchFacesRequest);
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face : faceImageMatches) {
                System.out.println("The similarity level is  " +
 face.similarity());
                System.out.println();
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考[SearchFacesByImage](#)中的。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
```

```
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id,
                Image=image.image,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            image_face = RekognitionFace(
                {
                    "BoundingBox": response["SearchedFaceBoundingBox"],
                    "Confidence": response["SearchedFaceConfidence"],
                }
            )
            collection_faces = [
                RekognitionFace(face["Face"]) for face in response["FaceMatches"]
            ]
            logger.info(
                "Found %s faces in the collection that match the largest "
                "face in %s.",
                len(collection_faces),
                image.image_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
```

```
                self.collection_id,
                image.image_name,
            )
            raise
        else:
            return image_face, collection_faces
```

- 有关 API 的详细信息，请参阅适用SearchFacesByImage于 Python 的AWS SDK (Boto3) API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

亚马逊 Rekognition 使用场景 AWS SDKs

以下代码示例向您展示了如何使用在 Amazon Rekognition 中实现常见场景。 AWS SDKs这些场景向您展示了如何通过调用 Amazon Rekognition 中的多个函数或与其他 AWS 服务结合来完成特定任务。每个场景都包含完整源代码的链接，您可以在其中找到有关如何设置和运行代码的说明。

场景以中等水平的经验为目标，可帮助您结合具体环境了解服务操作。

示例

- 构建 Amazon Rekognition 收藏并使用软件开发工具包在其中寻找面孔 AWS
- 创建照片资产管理应用程序，让用户能够使用标签管理照片
- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的个人防护装备 AWS
- 使用软件开发工具包使用 Amazon Rekognition 检测和显示图像中的元素 AWS
- 使用 AWS SDK 检测图像中的人脸
- 使用亚马逊 Rekognition 和软件开发工具包检测视频中的信息 AWS
- 使用软件开发工具包使用 Amazon Rekognition 检测图像中的物体 AWS
- 使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS
- 使用 SDK 保存 EXIF 和其他图像信息 AWS

构建 Amazon Rekognition 收藏并使用软件开发工具包在其中寻找面孔 AWS

以下代码示例展示了如何：

- 创建 Amazon Rekognition 集合。
- 将图像添加到集合中并检测其中的人脸。
- 在集合中搜索与参照图像相匹配的人脸。
- 删除集合。

有关更多信息，请参阅搜索集合中的人脸。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建包装 Amazon Rekognition 函数的类。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
from rekognition_objects import RekognitionFace
from rekognition_image_detection import RekognitionImage

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
```

```
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened
 and its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from
 the
                 file.
        """
        with open(image_file_name, "rb") as img_file:
            image = {"Bytes": img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)


class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition
 API.
    """

    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client
```

```
    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id
            )
            response["CollectionId"] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection


    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({"CollectionId": col_id},
self.rekognition_client)
                for col_id in response["CollectionIds"]
            ]
        except ClientError:
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections
```

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection["CollectionId"]
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection
        )
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get("CollectionArn"),
            collection.get("FaceCount", 0),
            collection.get("CreationTimestamp"),
        )


    def to_dict(self):
        """
        Renders parts of the collection data to a dict.

        :return: The collection data as a dict.
        """
```

```
        rendering = {
            "collection_id": self.collection_id,
            "collection_arn": self.collection_arn,
            "face_count": self.face_count,
            "created": self.created,
        }
        return rendering


    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id
            )
            # Work around capitalization of Arn vs. ARN
            response["CollectionArn"] = response.get("CollectionARN")
            (
                self.collection_arn,
                self.face_count,
                self.created,
            ) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
self.collection_id)
            raise
        else:
            return self.to_dict()


    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
```

```
        except ClientError:
            logger.exception("Couldn't delete collection %s.",
self.collection_id)
            raise


    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id,
                Image=image.image,
                ExternalImageId=image.image_name,
                MaxFaces=max_faces,
                DetectionAttributes=["ALL"],
            )
            indexed_faces = [
                RekognitionFace({**face["Face"], **face["FaceDetail"]})
                for face in response["FaceRecords"]
            ]
            unindexed_faces = [
                RekognitionFace(face["FaceDetail"])
                for face in response["UnindexedFaces"]
            ]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
                len(indexed_faces),
                image.image_name,
                len(unindexed_faces),
            )
        except ClientError:
            logger.exception("Couldn't index faces in image %s.",
image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces
```

```
    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results
            )
            faces = [RekognitionFace(face) for face in response["Faces"]]
            logger.info(
                "Found %s faces in collection %s.", len(faces),
self.collection_id
            )
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id
            )
            raise
        else:
            return faces


    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list
does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id,
                FaceId=face_id,
```

```
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            faces = [RekognitionFace(face["Face"]) for face in
response["FaceMatches"]]
            logger.info(
                "Found %s faces in %s that match %s.",
                len(faces),
                self.collection_id,
                face_id,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                face_id,
            )
            raise
        else:
            return faces


    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id,
                Image=image.image,
                FaceMatchThreshold=threshold,
                MaxFaces=max_faces,
            )
            image_face = RekognitionFace(
```

```
                {
                    "BoundingBox": response["SearchedFaceBoundingBox"],
                    "Confidence": response["SearchedFaceConfidence"],
                }
            )
            collection_faces = [
                RekognitionFace(face["Face"]) for face in response["FaceMatches"]
            ]
            logger.info(
                "Found %s faces in the collection that match the largest "
                "face in %s.",
                len(collection_faces),
                image.image_name,
            )
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
                self.collection_id,
                image.image_name,
            )
            raise
        else:
            return image_face, collection_faces


class RekognitionFace:
    """Encapsulates an Amazon Rekognition face."""

    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get("BoundingBox")
        self.confidence = face.get("Confidence")
        self.landmarks = face.get("Landmarks")
        self.pose = face.get("Pose")
        self.quality = face.get("Quality")
        age_range = face.get("AgeRange")
        if age_range is not None:
```

```
            self.age_range = (age_range.get("Low"), age_range.get("High"))
        else:
            self.age_range = None
        self.smile = face.get("Smile", {}).get("Value")
        self.eyeglasses = face.get("Eyeglasses", {}).get("Value")
        self.sunglasses = face.get("Sunglasses", {}).get("Value")
        self.gender = face.get("Gender", {}).get("Value", None)
        self.beard = face.get("Beard", {}).get("Value")
        self.mustache = face.get("Mustache", {}).get("Value")
        self.eyes_open = face.get("EyesOpen", {}).get("Value")
        self.mouth_open = face.get("MouthOpen", {}).get("Value")
        self.emotions = [
            emo.get("Type")
            for emo in face.get("Emotions", [])
            if emo.get("Confidence", 0) > 50
        ]
        self.face_id = face.get("FaceId")
        self.image_id = face.get("ImageId")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
        rendering = {}
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.age_range is not None:
            rendering["age"] = f"{self.age_range[0]} - {self.age_range[1]}"
        if self.gender is not None:
            rendering["gender"] = self.gender
        if self.emotions:
            rendering["emotions"] = self.emotions
        if self.face_id is not None:
            rendering["face_id"] = self.face_id
        if self.image_id is not None:
            rendering["image_id"] = self.image_id
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        has = []
        if self.smile:
            has.append("smile")
```

```
        if self.eyeglasses:
            has.append("eyeglasses")
        if self.sunglasses:
            has.append("sunglasses")
        if self.beard:
            has.append("beard")
        if self.mustache:
            has.append("mustache")
        if self.eyes_open:
            has.append("open eyes")
        if self.mouth_open:
            has.append("open mouth")
        if has:
            rendering["has"] = has
        return rendering
```

使用包装类从一组图像中构建人脸集合，然后在集合中搜索人脸。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition face collection demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    rekognition_client = boto3.client("rekognition")
    images = [
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128316.jpg",
            rekognition_client,
            image_name="sitting",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128317.jpg",
            rekognition_client,
            image_name="hopping",
        ),
        RekognitionImage.from_file(
            ".media/pexels-agung-pandit-wiguna-1128318.jpg",
            rekognition_client,
```

```
            image_name="biking",
        ),
    ]

    collection_mgr = RekognitionCollectionManager(rekognition_client)
    collection = collection_mgr.create_collection("doc-example-collection-demo")
    print(f"Created collection {collection.collection_id}:")
    pprint(collection.describe_collection())

    print("Indexing faces from three images:")
    for image in images:
        collection.index_faces(image, 10)
    print("Listing faces in collection:")
    faces = collection.list_faces(10)
    for face in faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(
        f"Searching for faces in the collection that match the first face in the
"
        f"list (Face ID: {faces[0].face_id}."
    )
    found_faces = collection.search_faces(faces[0].face_id, 80, 10)
    print(f"Found {len(found_faces)} matching faces.")
    for face in found_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(
        f"Searching for faces in the collection that match the largest face in "
        f"{images[0].image_name}."
    )
    image_face, match_faces = collection.search_faces_by_image(images[0], 80, 10)
    print(f"The largest face in {images[0].image_name} is:")
    pprint(image_face.to_dict())
    print(f"Found {len(match_faces)} matching faces.")
    for face in match_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    collection.delete_collection()
    print("Thanks for watching!")
    print("-" * 88)
```

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

创建照片资产管理应用程序，让用户能够使用标签管理照片

以下代码示例演示了如何创建无服务器应用程序，让用户能够使用标签管理照片。

.NET

适用于 .NET 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

适用于 Java 的 SDK 2.x

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

适用于 Kotlin 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

适用于 PHP 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

适用于 Rust 的 SDK

演示如何开发照片资产管理应用程序，该应用程序使用 Amazon Rekognition 检测图像中的标签并将其存储以供日后检索。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例 GitHub。

要深入了解这个例子的起源，请参阅 AWS 社区上的博文。

本示例中使用的服务

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包使用 Amazon Rekognition 检测图像中的个人防护装备 AWS

以下代码示例展示如何构建采用 Amazon Rekognition 来检测图像中的个人防护设备（PPE）的应用程序。

Java

适用于 Java 的 SDK 2.x

演示如何创建使用个人防护设备检测图像的 AWS Lambda 功能。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包使用 Amazon Rekognition 检测和显示图像中的元素 AWS

以下代码示例展示了如何：

- 使用 Amazon Rekognition 检测图像中的元素。
- 显示图像并在检测到的元素周围绘制边界框。

有关更多信息，请参阅显示边界框。

Python

适用于 Python 的 SDK（Boto3）

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建类来包装 Amazon Rekognition 函数。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
import requests

from rekognition_objects import (
    RekognitionFace,
    RekognitionCelebrity,
    RekognitionLabel,
    RekognitionModerationLabel,
    RekognitionText,
    show_bounding_boxes,
    show_polygons,
)

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """

    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
```

```
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client


    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened
and its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from
the
                 file.
        """
        with open(image_file_name, "rb") as img_file:
            image = {"Bytes": img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)


    @classmethod
    def from_bucket(cls, s3_object, rekognition_client):
        """
        Creates a RekognitionImage object from an Amazon S3 object.

        :param s3_object: An Amazon S3 object that identifies the image. The
image
                          is not retrieved until needed for a later call.
        :param rekognition_client: A Boto3 Rekognition client.
        :return: The RekognitionImage object, initialized with Amazon S3 object
data.
        """
        image = {"S3Object": {"Bucket": s3_object.bucket_name, "Name":
s3_object.key}}
        return cls(image, s3_object.key, rekognition_client)
```

```
    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=["ALL"]
            )
            faces = [RekognitionFace(face) for face in response["FaceDetails"]]
            logger.info("Detected %s faces.", len(faces))
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces


    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels
            )
            labels = [RekognitionLabel(label) for label in response["Labels"]]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels


    def recognize_celebrities(self):
        """
        Detects celebrities in the image.
```

```
        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response =
self.rekognition_client.recognize_celebrities(Image=self.image)
            celebrities = [
                RekognitionCelebrity(celeb) for celeb in
response["CelebrityFaces"]
            ]
            other_faces = [
                RekognitionFace(face) for face in response["UnrecognizedFaces"]
            ]
            logger.info(
                "Found %s celebrities and %s other faces in %s.",
                len(celebrities),
                len(other_faces),
                self.image_name,
            )
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.",
self.image_name)
            raise
        else:
            return celebrities, other_faces



    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value
greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that
have
                 a similarity value below the specified threshold.
        """
        try:
```

```
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity,
            )
            matches = [
                RekognitionFace(match["Face"]) for match in
response["FaceMatches"]
            ]
            unmatches = [RekognitionFace(face) for face in
response["UnmatchedFaces"]]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches),
                len(unmatches),
            )
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.",
                self.image_name,
                target_image.image_name,
            )
            raise
        else:
            return matches, unmatches


    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify
content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image
            )
            labels = [
                RekognitionModerationLabel(label)
                for label in response["ModerationLabels"]
            ]
            logger.info(
```

```
                "Found %s moderation labels in %s.", len(labels), self.image_name
            )
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name
            )
            raise
        else:
            return labels


    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in
 response["TextDetections"]]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

创建辅助函数来绘制边界框和多边形。

```
import io
import logging
from PIL import Image, ImageDraw

logger = logging.getLogger(__name__)


def show_bounding_boxes(image_bytes, box_sets, colors):
    """
    Draws bounding boxes on an image and shows it with the default image viewer.
```

```
    :param image_bytes: The image to draw, as bytes.
    :param box_sets: A list of lists of bounding boxes to draw on the image.
    :param colors: A list of colors to use to draw the bounding boxes.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for boxes, color in zip(box_sets, colors):
        for box in boxes:
            left = image.width * box["Left"]
            top = image.height * box["Top"]
            right = (image.width * box["Width"]) + left
            bottom = (image.height * box["Height"]) + top
            draw.rectangle([left, top, right, bottom], outline=color, width=3)
    image.show()



def show_polygons(image_bytes, polygons, color):
    """
    Draws polygons on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param polygons: The list of polygons to draw on the image.
    :param color: The color to use to draw the polygons.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for polygon in polygons:
        draw.polygon(
            [
                (image.width * point["X"], image.height * point["Y"])
                for point in polygon
            ],
            outline=color,
        )
    image.show()
```

创建类来解析 Amazon Rekognition 返回的对象。

```
class RekognitionFace:
```

```
    """Encapsulates an Amazon Rekognition face."""

    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get("BoundingBox")
        self.confidence = face.get("Confidence")
        self.landmarks = face.get("Landmarks")
        self.pose = face.get("Pose")
        self.quality = face.get("Quality")
        age_range = face.get("AgeRange")
        if age_range is not None:
            self.age_range = (age_range.get("Low"), age_range.get("High"))
        else:
            self.age_range = None
        self.smile = face.get("Smile", {}).get("Value")
        self.eyeglasses = face.get("Eyeglasses", {}).get("Value")
        self.sunglasses = face.get("Sunglasses", {}).get("Value")
        self.gender = face.get("Gender", {}).get("Value", None)
        self.beard = face.get("Beard", {}).get("Value")
        self.mustache = face.get("Mustache", {}).get("Value")
        self.eyes_open = face.get("EyesOpen", {}).get("Value")
        self.mouth_open = face.get("MouthOpen", {}).get("Value")
        self.emotions = [
            emo.get("Type")
            for emo in face.get("Emotions", [])
            if emo.get("Confidence", 0) > 50
        ]
        self.face_id = face.get("FaceId")
        self.image_id = face.get("ImageId")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
```

```
        rendering = {}
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.age_range is not None:
            rendering["age"] = f"{self.age_range[0]} - {self.age_range[1]}"
        if self.gender is not None:
            rendering["gender"] = self.gender
        if self.emotions:
            rendering["emotions"] = self.emotions
        if self.face_id is not None:
            rendering["face_id"] = self.face_id
        if self.image_id is not None:
            rendering["image_id"] = self.image_id
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        has = []
        if self.smile:
            has.append("smile")
        if self.eyeglasses:
            has.append("eyeglasses")
        if self.sunglasses:
            has.append("sunglasses")
        if self.beard:
            has.append("beard")
        if self.mustache:
            has.append("mustache")
        if self.eyes_open:
            has.append("open eyes")
        if self.mouth_open:
            has.append("open mouth")
        if has:
            rendering["has"] = has
        return rendering



class RekognitionCelebrity:
    """Encapsulates an Amazon Rekognition celebrity."""

    def __init__(self, celebrity, timestamp=None):
        """
        Initializes the celebrity object.
```

```
        :param celebrity: Celebrity data, in the format returned by Amazon
 Rekognition
                          functions.
        :param timestamp: The time when the celebrity was detected, if the
 celebrity
                          was detected in a video.
        """
        self.info_urls = celebrity.get("Urls")
        self.name = celebrity.get("Name")
        self.id = celebrity.get("Id")
        self.face = RekognitionFace(celebrity.get("Face"))
        self.confidence = celebrity.get("MatchConfidence")
        self.bounding_box = celebrity.get("BoundingBox")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the celebrity data to a dict.

        :return: A dict that contains the celebrity data.
        """
        rendering = self.face.to_dict()
        if self.name is not None:
            rendering["name"] = self.name
        if self.info_urls:
            rendering["info URLs"] = self.info_urls
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionPerson:
    """Encapsulates an Amazon Rekognition person."""

    def __init__(self, person, timestamp=None):
        """
        Initializes the person object.

        :param person: Person data, in the format returned by Amazon Rekognition
                       functions.
        :param timestamp: The time when the person was detected, if the person
                          was detected in a video.
        """
```

```
        self.index = person.get("Index")
        self.bounding_box = person.get("BoundingBox")
        face = person.get("Face")
        self.face = RekognitionFace(face) if face is not None else None
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the person data to a dict.

        :return: A dict that contains the person data.
        """
        rendering = self.face.to_dict() if self.face is not None else {}
        if self.index is not None:
            rendering["index"] = self.index
        if self.bounding_box is not None:
            rendering["bounding_box"] = self.bounding_box
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionLabel:
    """Encapsulates an Amazon Rekognition label."""

    def __init__(self, label, timestamp=None):
        """
        Initializes the label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the label was detected, if the label
                          was detected in a video.
        """
        self.name = label.get("Name")
        self.confidence = label.get("Confidence")
        self.instances = label.get("Instances")
        self.parents = label.get("Parents")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the label data to a dict.
```

```
        :return: A dict that contains the label data.
        """
        rendering = {}
        if self.name is not None:
            rendering["name"] = self.name
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering



class RekognitionModerationLabel:
    """Encapsulates an Amazon Rekognition moderation label."""

    def __init__(self, label, timestamp=None):
        """
        Initializes the moderation label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the moderation label was detected, if the
                          label was detected in a video.
        """
        self.name = label.get("Name")
        self.confidence = label.get("Confidence")
        self.parent_name = label.get("ParentName")
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the moderation label data to a dict.

        :return: A dict that contains the moderation label data.
        """
        rendering = {}
        if self.name is not None:
            rendering["name"] = self.name
        if self.parent_name is not None:
            rendering["parent_name"] = self.parent_name
        if self.timestamp is not None:
            rendering["timestamp"] = self.timestamp
        return rendering
```

```
class RekognitionText:
    """Encapsulates an Amazon Rekognition text element."""

    def __init__(self, text_data):
        """
        Initializes the text object.

        :param text_data: Text data, in the format returned by Amazon Rekognition
                          functions.
        """
        self.text = text_data.get("DetectedText")
        self.kind = text_data.get("Type")
        self.id = text_data.get("Id")
        self.parent_id = text_data.get("ParentId")
        self.confidence = text_data.get("Confidence")
        self.geometry = text_data.get("Geometry")

    def to_dict(self):
        """
        Renders some of the text data to a dict.

        :return: A dict that contains the text data.
        """
        rendering = {}
        if self.text is not None:
            rendering["text"] = self.text
        if self.kind is not None:
            rendering["kind"] = self.kind
        if self.geometry is not None:
            rendering["polygon"] = self.geometry.get("Polygon")
        return rendering
```

使用包装类来检测图像中的元素并显示其边界框。可以在上找到本示例中使用的图像 GitHub 以及说明和更多代码。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Rekognition image detection demo!")
```

```
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    rekognition_client = boto3.client("rekognition")
    street_scene_file_name = ".media/pexels-kaique-rocha-109919.jpg"
    celebrity_file_name = ".media/pexels-pixabay-53370.jpg"
    one_girl_url = "https://dhei5unw3vrsx.cloudfront.net/images/
source3_resized.jpg"
    three_girls_url = "https://dhei5unw3vrsx.cloudfront.net/images/
target3_resized.jpg"
    swimwear_object = boto3.resource("s3").Object(
        "console-sample-images-pdx", "yoga_swimwear.jpg"
    )
    book_file_name = ".media/pexels-christina-morillo-1181671.jpg"

    street_scene_image = RekognitionImage.from_file(
        street_scene_file_name, rekognition_client
    )
    print(f"Detecting faces in {street_scene_image.image_name}...")
    faces = street_scene_image.detect_faces()
    print(f"Found {len(faces)} faces, here are the first three.")
    for face in faces[:3]:
        pprint(face.to_dict())
    show_bounding_boxes(
        street_scene_image.image["Bytes"],
        [[face.bounding_box for face in faces]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    print(f"Detecting labels in {street_scene_image.image_name}...")
    labels = street_scene_image.detect_labels(100)
    print(f"Found {len(labels)} labels.")
    for label in labels:
        pprint(label.to_dict())
    names = []
    box_sets = []
    colors = ["aqua", "red", "white", "blue", "yellow", "green"]
    for label in labels:
        if label.instances:
            names.append(label.name)
            box_sets.append([inst["BoundingBox"] for inst in label.instances])
    print(f"Showing bounding boxes for {names} in {colors[:len(names)]}.")
    show_bounding_boxes(
```

```
        street_scene_image.image["Bytes"], box_sets, colors[: len(names)]
    )
    input("Press Enter to continue.")

    celebrity_image = RekognitionImage.from_file(
        celebrity_file_name, rekognition_client
    )
    print(f"Detecting celebrities in {celebrity_image.image_name}...")
    celebs, others = celebrity_image.recognize_celebrities()
    print(f"Found {len(celebs)} celebrities.")
    for celeb in celebs:
        pprint(celeb.to_dict())
    show_bounding_boxes(
        celebrity_image.image["Bytes"],
        [[celeb.face.bounding_box for celeb in celebs]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    girl_image_response = requests.get(one_girl_url)
    girl_image = RekognitionImage(
        {"Bytes": girl_image_response.content}, "one-girl", rekognition_client
    )
    group_image_response = requests.get(three_girls_url)
    group_image = RekognitionImage(
        {"Bytes": group_image_response.content}, "three-girls",
rekognition_client
    )
    print("Comparing reference face to group of faces...")
    matches, unmatches = girl_image.compare_faces(group_image, 80)
    print(f"Found {len(matches)} face matching the reference face.")
    show_bounding_boxes(
        group_image.image["Bytes"],
        [[match.bounding_box for match in matches]],
        ["aqua"],
    )
    input("Press Enter to continue.")

    swimwear_image = RekognitionImage.from_bucket(swimwear_object,
rekognition_client)
    print(f"Detecting suggestive content in {swimwear_object.key}...")
    labels = swimwear_image.detect_moderation_labels()
    print(f"Found {len(labels)} moderation labels.")
    for label in labels:
```

```
        pprint(label.to_dict())
    input("Press Enter to continue.")

    book_image = RekognitionImage.from_file(book_file_name, rekognition_client)
    print(f"Detecting text in {book_image.image_name}...")
    texts = book_image.detect_text()
    print(f"Found {len(texts)} text instances. Here are the first seven:")
    for text in texts[:7]:
        pprint(text.to_dict())
    show_polygons(
        book_image.image["Bytes"], [text.geometry["Polygon"] for text in texts],
"aqua"
    )

    print("Thanks for watching!")
    print("-" * 88)
```

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 AWS SDK 检测图像中的人脸

以下代码示例展示了如何：

- 将图像保存到 Amazon S3 存储桶中。
- 使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（如微笑）。
- 显示这些细节。

Rust

适用于 Rust 的 SDK

将图像保存到具有 uploads 前缀的 Amazon S3 存储桶中，使用 Amazon Rekognition 检测面部细节，例如年龄范围、性别和情绪（微笑等），并显示这些细节。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用亚马逊 Rekognition 和软件开发工具包检测视频中的信息 AWS

以下代码示例演示了如何：

- 启动 Amazon Rekognition 任务，检测视频中的人物、对象和文本等元素。
- 查看任务状态，直到任务完成。
- 输出每个任务检测到的元素列表。

Java

适用于 Java 的 SDK 2.x

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

从位于 Amazon S3 存储桶中的视频获取名人结果。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
```

```
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;

/**
 * To run this code example, ensure that you perform the Prerequisites as stated
 * in the Amazon Rekognition Guide:
 * https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
 *
 * Also, ensure that set up your development environment, including your
 * credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class VideoCelebrityDetection {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startCelebrityDetection(rekClient, channel, bucket, video);
        getCelebrityDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startCelebrityDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartCelebrityRecognitionRequest recognitionRequest =
StartCelebrityRecognitionRequest.builder()
                    .jobTag("Celebrities")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();
```

```
            StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
rekClient
                    .startCelebrityRecognition(recognitionRequest);
            startJobId = startCelebrityRecognitionResult.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getCelebrityDetectionResults(RekognitionClient rekClient)
{
        try {
            String paginationToken = null;
            GetCelebrityRecognitionResponse recognitionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (recognitionResponse != null)
                    paginationToken = recognitionResponse.nextToken();

                GetCelebrityRecognitionRequest recognitionRequest =
GetCelebrityRecognitionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds
                while (!finished) {
                    recognitionResponse =
rekClient.getCelebrityRecognition(recognitionRequest);
                    status = recognitionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
```

```
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 recognitionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<CelebrityRecognition> celebs =
 recognitionResponse.celebrities();
                for (CelebrityRecognition celeb : celebs) {
                    long seconds = celeb.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    CelebrityDetail details = celeb.celebrity();
                    System.out.println("Name: " + details.name());
                    System.out.println("Id: " + details.id());
                    System.out.println();
                }

            } while (recognitionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

通过标签检测操作检测视频中的标签。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
```

```
                    bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                    video - The name of the video (for example, people.mp4).\s
                    queueUrl- The URL of a SQS queue.\s
                    topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    public static void startLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
```

```
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .minConfidence(50F)
                    .build();

            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .maxResults(10)
                        .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy + " status is: " + status);

                Thread.sleep(1000);
```

```
                yy++;
            }

            System.out.println(startJobId + " status is: " + status);

        } catch (RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message : messages) {
                    String notification = message.body();

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId) == 0) {
                        System.out.println("Job id: " + operationJobId);
```

```
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            getResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    } else {
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void getResultsLabels(RekognitionClient rekClient) {

        int maxResults = 10;
        String paginationToken = null;
        GetLabelDetectionResponse labelDetectionResult = null;

        try {
            do {
                if (labelDetectionResult != null)
                    paginationToken = labelDetectionResult.nextToken();

                GetLabelDetectionRequest labelDetectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .sortBy(LabelDetectionSortBy.TIMESTAMP)
                        .maxResults(maxResults)
```

```
                        .nextToken(paginationToken)
                        .build();

                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData =
labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels =
labelDetectionResult.labels();
                for (LabelDetection detectedLabel : detectedLabels) {
                    long seconds = detectedLabel.timestamp();
                    Label label = detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
instance.confidence().toString());
                            System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
```

```
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult != null &&
 labelDetectionResult.nextToken() != null);

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的人脸

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
```

```
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of the video (for example, people.mp4).\s
                   queueUrl- The URL of a SQS queue.\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    public static void startLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .minConfidence(50F)
                    .build();
```

```
            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .maxResults(10)
                        .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy + " status is: " + status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId + " status is: " + status);

        } catch (RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {
        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
                .queueUrl(queueUrl)
                .build();

        try {
```

```
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message : messages) {
                    String notification = message.body();

                    // Get the status and job id from the notification
                    ObjectMapper mapper = new ObjectMapper();
                    JsonNode jsonMessageTree = mapper.readTree(notification);
                    JsonNode messageBodyText = jsonMessageTree.get("Message");
                    ObjectMapper operationResultMapper = new ObjectMapper();
                    JsonNode jsonResultTree =
operationResultMapper.readTree(messageBodyText.textValue());
                    JsonNode operationJobId = jsonResultTree.get("JobId");
                    JsonNode operationStatus = jsonResultTree.get("Status");
                    System.out.println("Job found in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId) == 0) {
                        System.out.println("Job id: " + operationJobId);
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            getResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    } else {
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch (RekognitionException e) {
            e.getMessage();
```

```
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void getResultsLabels(RekognitionClient rekClient) {

        int maxResults = 10;
        String paginationToken = null;
        GetLabelDetectionResponse labelDetectionResult = null;

        try {
            do {
                if (labelDetectionResult != null)
                    paginationToken = labelDetectionResult.nextToken();

                GetLabelDetectionRequest labelDetectionRequest =
GetLabelDetectionRequest.builder()
                        .jobId(startJobId)
                        .sortBy(LabelDetectionSortBy.TIMESTAMP)
                        .maxResults(maxResults)
                        .nextToken(paginationToken)
                        .build();

                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData =
labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels =
labelDetectionResult.labels();
                for (LabelDetection detectedLabel : detectedLabels) {
                    long seconds = detectedLabel.timestamp();
                    Label label = detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");
```

```
                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
 detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
 instance.confidence().toString());
                            System.out.println("        Bounding box: " +
 instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
 ":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult != null &&
 labelDetectionResult.nextToken() != null);

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的不当或冒犯性内容。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
```

```
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startModerationDetection(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {

        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();
```

```
            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                    .jobTag("Moderation")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartContentModerationResponse startModDetectionResult = rekClient
                    .startContentModeration(modDetectionRequest);
            startJobId = startModDetectionResult.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getModResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetContentModerationResponse modDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (modDetectionResponse != null)
                    paginationToken = modDetectionResponse.nextToken();

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
```

```
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
 null.
                VideoMetadata videoMetaData =
 modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
 videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<ContentModerationDetection> mods =
 modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod : mods) {
                    long seconds = mod.timestamp() / 1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }

            } while (modDetectionResponse != null &&
 modDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的技术提示片段和镜头检测片段。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartShotDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartTechnicalCueDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionFilters;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.SegmentDetection;
import software.amazon.awssdk.services.rekognition.model.TechnicalCueSegment;
import software.amazon.awssdk.services.rekognition.model.ShotSegment;
import software.amazon.awssdk.services.rekognition.model.SegmentType;
import software.amazon.awssdk.services.sqs.SqsClient;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectSegment {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>
```

```
                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

       if (args.length != 4) {
           System.out.println(usage);
           System.exit(1);
       }

       String bucket = args[0];
       String video = args[1];
       String topicArn = args[2];
       String roleArn = args[3];

       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       SqsClient sqs = SqsClient.builder()
               .region(Region.US_EAST_1)
               .build();

       NotificationChannel channel = NotificationChannel.builder()
               .snsTopicArn(topicArn)
               .roleArn(roleArn)
               .build();

       startSegmentDetection(rekClient, channel, bucket, video);
       getSegmentResults(rekClient);
       System.out.println("This example is done!");
       sqs.close();
       rekClient.close();
   }

   public static void startSegmentDetection(RekognitionClient rekClient,
           NotificationChannel channel,
           String bucket,
```

```
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartShotDetectionFilter cueDetectionFilter =
StartShotDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();

            StartTechnicalCueDetectionFilter technicalCueDetectionFilter =
StartTechnicalCueDetectionFilter.builder()
                    .minSegmentConfidence(60F)
                    .build();

            StartSegmentDetectionFilters filters =
StartSegmentDetectionFilters.builder()
                    .shotFilter(cueDetectionFilter)
                    .technicalCueFilter(technicalCueDetectionFilter)
                    .build();

            StartSegmentDetectionRequest segDetectionRequest =
StartSegmentDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .segmentTypes(SegmentType.TECHNICAL_CUE, SegmentType.SHOT)
                    .video(vidOb)
                    .filters(filters)
                    .build();

            StartSegmentDetectionResponse segDetectionResponse =
rekClient.startSegmentDetection(segDetectionRequest);
            startJobId = segDetectionResponse.jobId();

        } catch (RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
```

```
    }

    public static void getSegmentResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetSegmentDetectionResponse segDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (segDetectionResponse != null)
                    paginationToken = segDetectionResponse.nextToken();

                GetSegmentDetectionRequest recognitionRequest =
GetSegmentDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    segDetectionResponse =
rekClient.getSegmentDetection(recognitionRequest);
                    status = segDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }
                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                List<VideoMetadata> videoMetaData =
segDetectionResponse.videoMetadata();
                for (VideoMetadata metaData : videoMetaData) {
                    System.out.println("Format: " + metaData.format());
                    System.out.println("Codec: " + metaData.codec());
```

```
                    System.out.println("Duration: " + metaData.durationMillis());
                    System.out.println("FrameRate: " + metaData.frameRate());
                    System.out.println("Job");
                }

                List<SegmentDetection> detectedSegments =
segDetectionResponse.segments();
                for (SegmentDetection detectedSegment : detectedSegments) {
                    String type = detectedSegment.type().toString();
                    if (type.contains(SegmentType.TECHNICAL_CUE.toString())) {
                        System.out.println("Technical Cue");
                        TechnicalCueSegment segmentCue =
detectedSegment.technicalCueSegment();
                        System.out.println("\tType: " + segmentCue.type());
                        System.out.println("\tConfidence: " +
segmentCue.confidence().toString());
                    }

                    if (type.contains(SegmentType.SHOT.toString())) {
                        System.out.println("Shot");
                        ShotSegment segmentShot = detectedSegment.shotSegment();
                        System.out.println("\tIndex " + segmentShot.index());
                        System.out.println("\tConfidence: " +
segmentShot.confidence().toString());
                    }

                    long seconds = detectedSegment.durationMillis();
                    System.out.println("\tDuration : " + seconds + "
milliseconds");
                    System.out.println("\tStart time code: " +
detectedSegment.startTimecodeSMPTE());
                    System.out.println("\tEnd time code: " +
detectedSegment.endTimecodeSMPTE());
                    System.out.println("\tDuration time code: " +
detectedSegment.durationSMPTE());
                    System.out.println();
                }

            } while (segDetectionResponse != null &&
segDetectionResponse.nextToken() != null);

        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
```

```
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的文本。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectText {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
```

```
                    bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                    video - The name of video (for example, people.mp4).\s
                    topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                 """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startTextLabels(rekClient, channel, bucket, video);
        getTextResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startTextLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();
```

```
            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
                    .jobTag("DetectingLabels")
                    .notificationChannel(channel)
                    .video(vidOb)
                    .build();

            StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getTextResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetTextDetectionResponse textDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (textDetectionResponse != null)
                    paginationToken = textDetectionResponse.nextToken();

                GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
```

```
                    textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                    status = textDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
textDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<TextDetectionResult> labels =
textDetectionResponse.textDetections();
                for (TextDetectionResult detectedText : labels) {
                    System.out.println("Confidence: " +
detectedText.textDetection().confidence().toString());
                    System.out.println("Id : " +
detectedText.textDetection().id());
                    System.out.println("Parent Id: " +
detectedText.textDetection().parentId());
                    System.out.println("Type: " +
detectedText.textDetection().type());
                    System.out.println("Text: " +
detectedText.textDetection().detectedText());
                    System.out.println();
                }

            } while (textDetectionResponse != null &&
textDetectionResponse.nextToken() != null);
```

```
        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的人物。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoPersonDetection {
    private static String startJobId = "";

    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    public static void startPersonLabels(RekognitionClient rekClient,
            NotificationChannel channel,
            String bucket,
            String video) {
        try {
```

```
            S3Object s3Obj = S3Object.builder()
                    .bucket(bucket)
                    .name(video)
                    .build();

            Video vidOb = Video.builder()
                    .s3Object(s3Obj)
                    .build();

            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                    .jobTag("DetectingLabels")
                    .video(vidOb)
                    .notificationChannel(channel)
                    .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getPersonDetectionResults(RekognitionClient rekClient) {
        try {
            String paginationToken = null;
            GetPersonTrackingResponse personTrackingResult = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (personTrackingResult != null)
                    paginationToken = personTrackingResult.nextToken();

                GetPersonTrackingRequest recognitionRequest =
GetPersonTrackingRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();
```

```
                // Wait until the job succeeds
                while (!finished) {

                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is
null.
                VideoMetadata videoMetaData =
personTrackingResult.videoMetadata();

                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " +
videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<PersonDetection> detectedPersons =
personTrackingResult.persons();
                for (PersonDetection detectedPerson : detectedPersons) {
                    long seconds = detectedPerson.timestamp() / 1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
detectedPerson.person().index());
                    System.out.println();
                }

            } while (personTrackingResult != null &&
personTrackingResult.nextToken() != null);
```

```
        } catch (RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API 参考》中的以下主题。
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

检测存储在 Amazon S3 存储桶内的视频中的人脸

```
suspend fun startFaceDetection(
    channelVal: NotificationChannel?,
    bucketVal: String,
```

```
    videoVal: String,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }

    val request =
        StartFaceDetectionRequest {
            jobTag = "Faces"
            faceAttributes = FaceAttributes.All
            notificationChannel = channelVal
            video = vidOb
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val startLabelDetectionResult = rekClient.startFaceDetection(request)
        startJobId = startLabelDetectionResult.jobId.toString()
    }
}

suspend fun getFaceResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        var response: GetFaceDetectionResponse? = null

        val recognitionRequest =
            GetFaceDetectionRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
            response = rekClient.getFaceDetection(recognitionRequest)
            status = response.jobStatus.toString()
            if (status.compareTo("Succeeded") == 0) {
```

```
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = response?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        // Show face information.
        response?.faces?.forEach { face ->
            println("Age: ${face.face?.ageRange}")
            println("Face: ${face.face?.beard}")
            println("Eye glasses: ${face?.face?.eyeglasses}")
            println("Mustache: ${face.face?.mustache}")
            println("Smile: ${face.face?.smile}")
        }
    }
}
```

检测存储在 Amazon S3 存储桶内的视频中的不当或冒犯性内容。

```
suspend fun startModerationDetection(
    channel: NotificationChannel?,
    bucketVal: String?,
    videoVal: String?,
) {
    val s3Obj =
        S3Object {
            bucket = bucketVal
            name = videoVal
        }
    val vidOb =
        Video {
            s3Object = s3Obj
        }
```

```
    val request =
        StartContentModerationRequest {
            jobTag = "Moderation"
            notificationChannel = channel
            video = vidOb
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val startModDetectionResult = rekClient.startContentModeration(request)
        startJobId = startModDetectionResult.jobId.toString()
    }
}

suspend fun getModResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var modDetectionResponse: GetContentModerationResponse? = null

        val modRequest =
            GetContentModerationRequest {
                jobId = startJobId
                maxResults = 10
            }

        // Wait until the job succeeds.
        while (!finished) {
            modDetectionResponse = rekClient.getContentModeration(modRequest)
            status = modDetectionResponse.jobStatus.toString()
            if (status.compareTo("Succeeded") == 0) {
                finished = true
            } else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = modDetectionResponse?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
```

```
        println("FrameRate: ${videoMetaData?.frameRate}")

        modDetectionResponse?.moderationLabels?.forEach { mod ->
            val seconds: Long = mod.timestamp / 1000
            print("Mod label: $seconds ")
            println(mod.moderationLabel)
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Kotlin API 参考》中的以下主题。
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用软件开发工具包使用 Amazon Rekognition 检测图像中的物体 AWS

以下代码示例展示如何构建采用 Amazon Rekognition 来按类别检测图像中对象的应用程序。

.NET

适用于 .NET 的 SDK

展示如何使用 Amazon Rekognition .NET API 创建应用程序，该应用程序采用 Amazon Rekognition 来按类别识别位于 Amazon Simple Storage Service (Amazon S3) 存储桶的图像中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

适用于 Java 的 SDK 2.x

展示如何使用 Amazon Rekognition Java API 创建应用程序，该应用程序采用 Amazon Rekognition 来按类别识别位于 Amazon Simple Storage Service (Amazon S3) 存储桶的图像当中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

演示如何使用 Amazon Rekogn 适用于 JavaScript 的 AWS SDK ition 和，创建一款应用程序，该应用程序使用 Amazon Rekognition 按类别识别位于亚马逊简单存储服务 (Amazon S3)

Simple S3 存储桶中的图像中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

了解如何：

- 使用 Amazon Cognito 创建未经身份验证的用户。
- 使用 Amazon Rekognition 分析包含对象的图像。
- 为 Amazon SES 验证电子邮件地址。
- 使用 Amazon SES 发送电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

适用于 Kotlin 的 SDK

展示如何使用 Amazon Rekognition Kotlin API 创建应用程序，该应用程序采用 Amazon Rekognition 来按类别识别位于 Amazon Simple Storage Service (Amazon S3) 存储桶的图像当中的对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

适用于 Python 的 SDK（Boto3）

向您展示如何使用创建 适用于 Python (Boto3) 的 AWS SDK 允许您执行以下操作的 Web 应用程序：

- 将照片上载到 Amazon Simple Storage Service (Amazon S3) 存储桶。
- 使用 Amazon Rekognition 来分析和标注照片。
- 使用 Amazon Simple Email Service (Amazon SES) 发送图像分析的电子邮件报告。

此示例包含两个主要组件：使用 React 构建的 JavaScript 网页和使用 Flask-RESTful 构建的用 Python 编写的 REST 服务。

可以使用 React 网页执行以下操作：

- 显示存储在 S3 存储桶中的图像列表。
- 将计算机中的图像上载到 S3 存储桶。
- 显示图像和用于识别图像中检测到的物品的标注。
- 获取 S3 存储桶中所有图像的报告并发送报告电子邮件。

该网页调用 REST 服务。该服务将请求发送到 AWS 以执行以下操作：

- 获取并筛选 S3 存储桶中的图像列表。
- 将照片上载到 S3 存储桶。
- 使用 Amazon Rekognition 分析各张照片并获取标注列表，这些标注用于识别在照片中检测到的物品。
- 分析 S3 存储桶中的所有照片，然后使用 Amazon SES 通过电子邮件发送报告。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 Amazon Rekognition 使用软件开发工具包检测视频中的人物和物体 AWS

以下代码示例展示如何使用 Amazon Rekognition 检测视频中的人物和对象。

Java

适用于 Java 的 SDK 2.x

展示如何使用 Amazon Rekognition Java API 创建应用程序，以检测位于 Amazon Simple Storage Service (Amazon S3) 存储桶的视频当中的人脸和对象。该应用程序使用 Amazon Simple Email Service (Amazon SES) 向管理员发送包含结果的电子邮件通知。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

适用于 Python 的 SDK（Boto3）

通过启动异步检测任务，使用 Amazon Rekognition 来检测视频中的人脸、对象和人物。此示例还将 Amazon Rekognition 配置为在任务完成时通知 Amazon Simple Notification Service (Amazon SNS) 主题，并订阅该主题的 Amazon Simple Queue Service (Amazon SQS) 队列。当队列收到有关任务的消息时，将检索该任务并输出结果。

最好在上查看此示例 GitHub。有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

使用 SDK 保存 EXIF 和其他图像信息 AWS

以下代码示例展示了如何：

- 从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息。
- 将图像文件上传到 Amazon S3 存储桶。
- 使用 Amazon Rekognition 识别文件中的三个主要属性（标签）。
- 将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

Rust

适用于 Rust 的 SDK

从 JPG、JPEG 或 PNG 文件中获取 EXIF 信息，将图像文件上传到 Amazon S3 存储桶，使用 Amazon Rekognition 识别文件中的三个主要属性（Amazon Rekognition 中的标签），然后将 EXIF 和标签信息添加到该区域的 Amazon DynamoDB 表中。

有关如何设置和运行的完整源代码和说明，请参阅上的完整示例GitHub。

本示例中使用的服务

- DynamoDB
- Amazon Rekognition
- Amazon S3

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将 Rekognition 与 SDK 配合使用 AWS。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

API 参考

Amazon Rekognition API 参考现位于 Amazon Rekognition API 参考。

Amazon Rekognition 安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以从专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构中受益。

使用以下主题了解如何保护您的 Amazon Rekognition 资源。

主题

- 适用于 Amazon Rekognition 的身份和访问管理
- Amazon Rekognition 中的数据保护
- 将 Amazon Rekognition 与 Amazon VPC 端点结合使用
- Amazon Rekognition 的合规性验证
- Amazon Rekognition 中的故障恢复能力
- Amazon Rekognition 中的配置和漏洞分析
- 防止跨服务混淆代理
- Amazon Rekognition 中的基础设施安全性

适用于 Amazon Rekognition 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和获得授权（具有权限）来使用 Amazon Rekognition 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon Rekognition 如何与 IAM 协同工作
- AWS 亚马逊 Rekognition 的托管政策
- 策略最佳实践
- 使用 Amazon Rekognition 控制台
- 使用 Amazon Rekognition 基于身份的策略示例

- Amazon Rekognition 基于资源的策略示例
- Amazon Rekognition 身份和访问问题排查

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在亚马逊 Rekognition 中所做的工作。

服务用户 - 如果您使用 Amazon Rekognition 服务来完成工作，您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon Rekognition 功能来完成工作，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon Rekognition 中的功能，请参阅Amazon Rekognition 身份和访问问题排查。

服务管理员 – 如果您在公司负责管理 Amazon Rekognition 资源，您可能对 Amazon Rekognition 具有完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon Rekognition 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Amazon Rekognition 搭配使用的更多信息，请参阅Amazon Rekognition 如何与 IAM 协同工作。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon Rekognition 的访问权限的详细信息。要查看您可在 IAM 中使用的 Amazon Rekognition 基于身份的策略示例，请参阅使用 Amazon Rekognition 基于身份的策略示例。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （ IAM Identity Center ）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的AWS 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的 IAM 中的AWS 多重身份验证。

IAM 用户和组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 AWS Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。

- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

使用基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

使用基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 AWS WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 AWS Organizations。 AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有

帐户。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。

- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 AWS 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 AWS 服务 该支持的列表 RCPs，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon Rekognition 如何与 IAM 协同工作

在使用 IAM 管理对 Amazon Rekognition 的访问权限之前，您应该了解哪些 IAM 功能可用于 Amazon Rekognition。要全面了解 Amazon Rekognition AWS 和其他服务如何与 IAM 配合使用 AWS ，请参阅 IAM 用户指南中的与 IAM 配合使用的服务。

主题

- Amazon Rekognition 基于身份的策略
- Amazon Rekognition 基于资源的策略
- Amazon Rekognition IAM 角色

Amazon Rekognition 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。Amazon Rekognition 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 Action 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

Amazon Rekognition 中的策略操作在操作前面使用以下前缀：rekognition:。例如，要授予某人使用 Amazon Rekognition DetectLabels API 操作在图像中检测目标、场景或概念的权限，您应将 rekognition:DetectLabels 操作纳入其策略中。策略语句必须包含 Action 或 NotAction 元素。Amazon Rekognition 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "rekognition:action1",
      "rekognition:action2"
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "rekognition:Describe*"
```

要查看 Amazon Rekognition 操作的列表，请参阅《IAM 用户指南》中的 Amazon Rekognition 定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

有关格式的更多信息 ARNs，请参阅 Amazon 资源名称 (ARNs) 和 AWS 服务命名空间。

例如，要在语句中指定 MyCollection 集合，请使用以下 ARN。

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/MyCollection"
```

要指定属于特定账户的所有实例，请使用通配符 (*)：

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/*"
```

无法对特定资源执行某些 Amazon Rekognition 操作，例如，用于创建资源的操作。在这些情况下，您必须使用通配符 （ *)。

```
"Resource": "*"
```

要查看 Amazon Rekognition 资源类型 ARNs及其列表，请参阅 IAM 用户指南中的亚马逊 Rekognition 定义的资源。要了解您可以使用哪些操作指定每个资源的 ARN，请参阅 Amazon Rekognition 定义的操作。

条件键

Amazon Rekognition 不提供任何特定于服务的条件键，但支持使用某些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

Amazon Rekognition 基于资源的策略

Amazon Rekognition 支持基于资源的自定义标签模型复制操作策略。有关更多信息，请参阅 Amazon Rekognition 基于资源的策略示例。

其他服务，如 Amazon S3，还支持基于资源的权限策略。例如，您可以将策略附加到 S3 存储桶以管理对该存储桶的访问权限。

要访问存储在 Amazon S3 存储桶中的图像，您必须有权访问 S3 存储桶中的对象。利用此权限，Amazon Rekognition 可从 S3 存储桶下载图像。以下示例策略允许用户对名为 amzn-s3-demo-bucket3 的 S3 存储桶执行s3:GetObject操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
}
```

要使用已启用版本控制的 S3 存储桶，请添加 `s3:GetObjectVersion` 操作，如以下示例所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
```

Amazon Rekognition IAM 角色

I AM 角色是您的 AWS 账户中具有特定权限的实体。

将临时凭证用于 Amazon Rekognition

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用AssumeRole或之类的 AWS STS API 操作来获取临时安全证书GetFederationToken。

Amazon Rekognition 支持使用临时凭证。

服务相关角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

Amazon Rekognition 不支持服务相关角色。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

Amazon Rekognition 支持服务角色。

使用服务角色可能会造成安全问题，即使用 Amazon Rekognition 调用其他服务，并对其不应该访问的资源采取行动。为了保护您的账户安全，您应将 Amazon Rekognition 的访问范围限制为仅限于您正在使用的资源。这可以通过将信任策略附加到您的 IAM 服务角色来完成。有关如何执行此操作的信息，请参阅 防止跨服务混淆代理。

在 Amazon Rekognition 中选择 IAM 角色

在您配置 Amazon Rekognition 来分析存储的视频时，您必须选择一个角色以允许 Amazon Rekognition 代表您访问 Amazon SNS。如果您之前已经创建了一个服务角色或服务相关角色，则 Amazon Rekognition 会为您提供一个角色列表供您选择。有关更多信息，请参阅 the section called “配置 Amazon Rekognition Video”。

示例：将 Amazon Rekognition 配置为访问亚马逊 S3 存储桶中的图像

以下是如何配置 Amazon Rekognition 以分析亚马逊 S3 存储桶中的图像的示例。如果您想使用 Amazon Rekognition 来分析 Amazon S3 存储桶中的图像，则必须执行以下操作：

1. 确保你的 IAM 用户/角色（客户端）有权调用相关的 Amazon Rekognition API 操作（比如等）DetectLabels DetectFaces

   附加基于身份的策略，该策略授予调用所需的 API 操作的相应权限。例如，要向您的角色授予调用`DetectLabels`和的权限`DetectFaces`，您需要为角色附加如下所示的策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:DetectLabels",
                "rekognition:DetectFaces"
```

```
                // other Rekognition permissions as needed
            ],
            "Resource": "*"
        }
    ]
}
```

2. 亚马逊 Rekognition 服务需要权限才能访问您的亚马逊 S3 存储桶。创建一个 IAM 服务角色，在调用 API 时，您需要将其传递给 Amazon Rekognition。确保服务角色：信任 Amazon Rekognition 服务委托人`s3:GetObject`，有权访问您的存储桶。

   信任策略可能如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   附加到服务角色的基于身份的策略可能如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS 亚马逊 Rekognition 的托管政策

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些政策涵盖常见用例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅 IAM 用户指南中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管式策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 托管式策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动一项新功能时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 IAM 用户指南中的适用于工作职能的AWS 托管式策略。

AWS 托管策略： AmazonRekognitionFullAccess

`AmazonRekognitionFullAccess` 授予对 Amazon Rekognition 资源的完全访问权，包括创建和删除集合。

您可以将 `AmazonRekognitionFullAccess` 策略附加到 IAM 身份。

权限详细信息

该策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

AWS 托管策略： AmazonRekognitionReadOnlyAccess

AmazonRekognitionReadOnlyAccess 授予对 Amazon Rekognition 资源的只读访问权限。

您可以将 AmazonRekognitionReadOnlyAccess 策略附加到 IAM 身份。

权限详细信息

该策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonRekognitionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage",
                "rekognition:DetectText",
                "rekognition:GetCelebrityInfo",
                "rekognition:RecognizeCelebrities",
                "rekognition:DetectModerationLabels",
                "rekognition:GetLabelDetection",
                "rekognition:GetFaceDetection",
                "rekognition:GetContentModeration",
                "rekognition:GetPersonTracking",
                "rekognition:GetCelebrityRecognition",
                "rekognition:GetFaceSearch",
                "rekognition:GetTextDetection",
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
```

```
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:ListProjectPolicies",
                "rekognition:ListUsers",
                "rekognition:SearchUsers",
                "rekognition:SearchUsersByImage",
                "rekognition:GetMediaAnalysisJob",
                "rekognition:ListMediaAnalysisJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AmazonRekognitionServiceRole

`AmazonRekognitionServiceRole` 允许 Amazon Rekognition 代表您调用 Amazon Kinesis Data Streams 和 Amazon SNS 服务。

您可以将 `AmazonRekognitionServiceRole` 策略附加到 IAM 身份。

如果使用此服务角色，则应将 Amazon Rekognition 的访问范围限制为仅限您正在使用的资源，从而保护您的账户安全。这可以通过将信任策略附加到您的 IAM 服务角色来完成。有关如何执行此操作的信息，请参阅 防止跨服务混淆代理。

权限详细信息

该策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:AmazonRekognition*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:*:*:stream/AmazonRekognition*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AmazonRekognitionCustomLabelsFullAccess

本策略适用于 Amazon Rekognition Custom Labels；用户。使用该 AmazonRekognitionCustomLabelsFullAccess 政策允许用户完全访问亚马逊 Rekognition 自定义标签 API，并完全访问由亚马逊 Rekognition 自定义标签控制台创建的控制台存储桶。

权限详细信息

该策略包含以下权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
```

```
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*custom-labels*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CopyProjectVersion",
                "rekognition:CreateProject",
                "rekognition:CreateProjectVersion",
                "rekognition:StartProjectVersion",
                "rekognition:StopProjectVersion",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DeleteProject",
                "rekognition:DeleteProjectVersion"
                "rekognition:TagResource",
                "rekognition:UntagResource",
                "rekognition:ListTagsForResource",
                "rekognition:CreateDataset",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:UpdateDatasetEntries",
                "rekognition:DistributeDatasetEntries",
                "rekognition:DeleteDataset",
                "rekognition:PutProjectPolicy",
                "rekognition:ListProjectPolicies",
                "rekognition:DeleteProjectPolicy"

            ],
            "Resource": "*"
        }
    ]
}
```

亚马逊 Rekognition 更新了托管政策 AWS

查看自该服务开始跟踪这些更改以来，Amazon Rekognition AWS 托管政策更新的详细信息。有关此页面更改的自动提示，请订阅 Amazon Rekognition 文档历史记录页面上的 RSS 源。

更改	描述	日期
涉及媒体分析工作的操作已添加到以下托管式策略中： • AWS 托管策略：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 在 AmazonRekognitionReadOnlyAccess 托管式策略中添加了以下操作： • GetMediaAnalysisJob • ListMediaAnalysisJob	2023 年 10 月 31 日
涉及管理用户的操作已添加到以下托管式策略中： • AWS 托管策略：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 在 AmazonRekognitionReadOnlyAccess 托管式策略中添加了以下操作： • ListUsers • SearchUsers • SearchUsersByImage	2023 年 6 月 12 日
针对 ProjectPolicy 和自定义标签模型复制的操作已添加到以下托管策略中： • AWS 托管策略：AmazonRekognitionFullAccess • AWS 托管策略：AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 在 AmazonRekognitionCustomLabelsFullAccess 和 AmazonRekognitionFullAccess 托管式策略中添加了以下操作： • CopyProjectVersion • PutProjectPolicy • ListProjectPolicies • DeleteProjectPolicy	2022 年 7 月 21 日
针对 ProjectPolicy 和自定义标签模型复制的操作已添加到以下托管策略中：	Amazon Rekognition 在 AmazonRekognitionReadOnlyAccess 托管式策略中添加了以下操作：	2022 年 7 月 21 日

更改	描述	日期
• AWS 托管策略：AmazonRekognitionReadOnlyAccess	• ListProjectPolicies	
以下托管式策略的数据集管理更新： • AWS 托管策略：AmazonRekognitionReadOnlyAccess • AWS 托管策略：AmazonRekognitionFullAccess • AWS 托管策略：AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 在、和托管策略中添加了以下操作 AmazonRekognitionReadOnlyAccess AmazonRekognitionFullOnlyAccess AmazonRekognitionCustomLabelsFullAccess • CreateDataset • ListDatasetEntries • ListDatasetLabels • DescribeDataset • UpdateDatasetEntries • DistributeDatasetEntries • DeleteDataset	2021 年 11 月 1 日
为 AWS 托管策略：AmazonRekognitionReadOnlyAccess 和 AWS 托管策略：AmazonRekognitionFullAccess 标记更新	Amazon Rekognition 在和政策中添加了新的标签操作。 AmazonRekognitionFullAccess AmazonRekognitionReadOnlyAccess	2021 年 4 月 2 日
Amazon Rekognition 已开始跟踪更改	亚马逊 Rekognition 开始跟踪其托管政策的变更。 AWS	2021 年 4 月 2 日

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon Rekognition 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管式策略或工作职能的AWS 托管式策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon Rekognition 控制台

除了 Amazon Rekognition Custom Labels 功能之外，Amazon Rekognition 在使用 Amazon Rekognition 控制台时不需要任何添加权限。有关 Amazon Rekognition Custom Labels 的信息，请参阅步骤 5：设置 Amazon Rekognition Custom Labels 控制台权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

使用 Amazon Rekognition 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon Rekognition 资源的权限。他们也无法使用 AWS Management Console AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

- 示例 Amazon Rekognition Custom Labels 策略
- 示例 1：允许用户对资源进行只读访问
- 示例 2：允许用户完全访问资源
- 允许用户查看他们自己的权限

示例 Amazon Rekognition Custom Labels 策略

您可以为 Amazon Rekognition Custom Labels 创建基于身份的策略。关更多信息，请参阅安全性。

示例 1：允许用户对资源进行只读访问

以下示例授予针对 Amazon Rekognition 资源的只读访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage",
                "rekognition:DetectText",
                "rekognition:GetCelebrityInfo",
```

```
                "rekognition:RecognizeCelebrities",
                "rekognition:DetectModerationLabels",
                "rekognition:GetLabelDetection",
                "rekognition:GetFaceDetection",
                "rekognition:GetContentModeration",
                "rekognition:GetPersonTracking",
                "rekognition:GetCelebrityRecognition",
                "rekognition:GetFaceSearch",
                "rekognition:GetTextDetection",
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
               "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset"

            ],
            "Resource": "*"
        }
    ]
}
```

示例 2：允许用户完全访问资源

以下示例授予针对 Amazon Rekognition 资源的完全访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        }
    ]
}
```

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Rekognition 基于资源的策略示例

Amazon Rekognition Custom Labels 使用基于资源的策略（称为项目策略）来管理模型版本的复制权限。

项目策略授予或拒绝将模型版本从源项目复制到目标项目的权限。如果目标项目位于不同的 AWS 账户中，或者您想限制账户内的访问权限，则需要项目策略。例如，您可能想要拒绝向特定 IAM 角色授予复制权限。 AWS 有关更多信息，请参阅复制模型。

授予权限以复制模型版本

以下示例允许主体 `arn:aws:iam::123456789012:role/Admin` 复制模型版本 `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080`。

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":{
        "AWS":"arn:aws:iam::123456789012:role/Admin"
      },
      "Action":"rekognition:CopyProjectVersion",
      "Resource":"arn:aws:rekognition:us-east-1:123456789012:project/my_project/
version/test_1/1627045542080"
    }
  ]
}
```

Amazon Rekognition 身份和访问问题排查

可以使用以下信息，以帮助您诊断和修复在使用 Amazon Rekognition 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon Rekognition 中执行操作
- 我无权执行 iam：PassRole
- 我是管理员并希望允许其他人访问 Amazon Rekognition
- 我想允许 AWS 账户以外的人访问我的亚马逊 Rekognition 资源

我无权在 Amazon Rekognition 中执行操作

如果 AWS Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是向您提供登录凭证的人。

如果 `mateojackson` IAM 用户尝试使用控制台查看有关 *`widget`* 的详细信息，但没有 `rekognition:`*`GetWidget`* 权限，则会出现以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rekognition:GetWidget on resource: my-example-widget
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `rekognition:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon Rekognition。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon Rekognition 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我是管理员并希望允许其他人访问 Amazon Rekognition

要允许其他人或应用程序访问 Amazon Rekognition，您必须向需要访问的人员或应用程序授予权限。如果使用 AWS IAM Identity Center 管理人员和应用程序，则可以向用户或组分配权限集来定义其访问权限级别。权限集会自动创建 IAM 策略并将其分配给与人员或应用程序关联的 IAM 角色。有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。

如果未使用 IAM Identity Center，则必须为需要访问的人员或应用程序创建 IAM 实体（用户或角色）。然后，您必须将策略附加到实体，以便在 Amazon Rekognition 中为它们授予正确的权限。授予权限后，向用户或应用程序开发人员提供凭证。他们将使用这些凭证访问 AWS。要了解有关创建 IAM 用户、组、策略和权限的更多信息，请参阅《IAM 用户指南》中的 IAM 身份和 IAM 中的策略和权限。

我想允许 AWS 账户以外的人访问我的亚马逊 Rekognition 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon Rekognition 是否支持这些功能，请参阅Amazon Rekognition 如何与 IAM 协同工作。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon Rekognition 中的数据保护

AWS 分担责任模型分担责任模型适用于 Amazon Rekognition 中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。

- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括你使用控制台、API 或与 Rekognition AWS 服务 或其他人合作时。 AWS CLI AWS SDKs在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

数据加密

以下信息说明 Amazon Rekognition 在何处使用数据加密来保护您的数据。

静态加密

Amazon Rekognition Image

映像

除非您通过访问 AI 服务选择退出策略页面并按照此处说明的流程选择退出，否则可能会存储传递给 Amazon Rekognition API 操作的图像并将其用于改进服务。使用 AWS Key Management Service (SSE-KMS) 对存储的图像进行静态加密 (Amazon S3)。

集合

对于在集合中存储信息的人脸比较操作，基础检测算法会先检测输入图像中的人脸，提取每个人脸的向量，然后将这些人脸向量存储在集合中。Amazon Rekognition 在执行人脸比较时使用这些人脸向量。人脸向量存储为浮点数组并进行静态加密。

Amazon Rekognition Video

视频

要分析视频，Amazon Rekognition 将视频复制到服务中进行处理。除非您通过访问 AI 服务选择退出策略页面并按照此处说明的流程选择退出，否则该视频可能会被存储并用于改进服务。使用 AWS Key Management Service (SSE-KMS) 对视频进行静态加密 (Amazon S3)。

Amazon Rekognition Custom Labels

Amazon Rekognition Custom Labels 加密您的静态数据。

映像

为训练您的模型，Amazon Rekognition Custom Labels 会复制您的源训练图像和测试图像。复制的图像在 Amazon Simple Storage Service (S3) 中使用服务器端加密，使用您提供的或 AWS 拥有的 K AWS KMS key MS 密钥进行静态加密。Amazon Rekognition Custom Labels 仅支持对称 KMS 密钥。您的源图像不受影响。有关更多信息，请参阅训练 Amazon Rekognition Custom Labels 模型。

模型

Amazon Rekognition Custom Labels 默认使用 AWS 拥有的密钥进行服务器端加密，加密存储在 Amazon S3 存储桶中的训练模型和清单文件。有关更多信息，请参阅使用服务器端加密保护数据。训练结果会写入到 CreateProjectVersion 的 `OutputConfig` 输入参数中指定的桶中。使用为存储桶（`OutputConfig`）配置的加密设置对训练结果进行加密。

控制台存储桶

Amazon Rekognition Custom Labels 控制台创建了一个可用于管理项目的 Amazon S3 存储桶（控制台存储桶）。控制台存储桶使用默认 Amazon S3 加密进行加密。有关更多信息，请参阅适用于 S3 存储桶的 Amazon Simple Storage Service 默认加密。如果您使用自己的 KMS 密钥，请在创建控制台存储桶后对其进行配置。有关更多信息，请参阅使用服务器端加密保护数据。Amazon Rekognition Custom Labels 会阻止公众访问控制台存储桶。

Rekognition Face Liveness

Rekognition Face Livenest 服务账户中存储的所有与会话相关的数据在静态状态下进行完全加密。默认情况下，参考和审核图像使用服务帐号中的 AWS 自有密钥进行加密。但是，您可以选择提供自己的 AWS KMS 密钥来加密这些图像。

传输中加密

Amazon Rekognition API 端点仅支持基于 HTTPS 的安全连接。所有通信都使用传输层安全性协议 (TLS) 进行加密。

密钥管理

您可以使用 AWS Key Management Service (SSE-KMS) 管理存储在 Amazon S3 存储桶中的输入图像和视频的密钥。有关更多信息，请参阅 AWS Key Management Service 概念。

面向人脸活跃度的客户管理密钥加密

CreateFaceLivenessSessionAPI 采用可选KmsKeyId参数。您可以提供您在账户中创建的 KMS 密钥的 id。此密钥将用于加密在 StartFaceLivenessSessionAPI 期间获得的参考和审计图像，在 GetFaceLivenessSessionResultsAPI 期间，将在返回结果之前使用此密钥对图像进行解密。如果 CreateFaceLivenessSession 请求包含 OutputConfig，则参考图像和审计图像将上传到指定的 Amazon S3 路径。我们建议在您的 Amazon S3 存储桶中启用服务器端加密 (SSE-S3)，以便数据在静态状态下继续保持加密状态。

当您提供自己的 AWS KMS 密钥 ID 时，Rekognition Face Liveness 服务将获得代表调用者的委托人使用客户托管密钥的权限。 APIs用于 APIs 从客户后端（APIsCreateFaceLivenessSession和）调用的委托人（用户或角色GetFaceLivenessSessionResults）必须有权执行以下操作：

- kms: DescribeKey
- kms: GenerateDataKey
- kms:Decrypt

互联网络流量隐私

Amazon Rekognition 的 Amazon Virtual Private Cloud (Amazon VPC) 端点是 VPC 内的逻辑实体，仅允许连接到 Amazon Rekognition。Amazon VPC 将请求路由到 Amazon Rekognition 并将响应路由回 VPC。有关更多信息，请参阅《Amazon VPC 用户指南》的 VPC 端点。有关将 Amazon VPC 端点与 Amazon Rekognition 配合使用的信息，请参阅 将 Amazon Rekognition 与 Amazon VPC 端点结合使用。

将 Amazon Rekognition 与 Amazon VPC 端点结合使用

如果您使用 Amazon Virtual Private Cloud (Amazon VPC) 托管 AWS 资源，则可以在您的 VPC 和 Amazon Rekognition 之间建立私有连接。您可以使用此连接实现 Amazon Rekognition 与 VPC 上的资源的通信而不用访问公共 Internet。

Amazon VPC 是一项 AWS 服务，可用来启动在虚拟网络中定义的 AWS 资源。借助 VPC，您可以控制您的网络设置，如 IP 地址范围、子网、路由表和网络网关。通过 VPC 端点，AWS 网络可处理 VPC 和 AWS 服务之间的路由。

要将您的 VPC 连接到 Amazon Rekognition，请为 Amazon Rekognition 定义一个接口 VPC 端点。接口终端节点是一个带有私有 IP 地址的 elastic network 接口，该地址用作发往受支持 AWS 服务的流量的入口点。该端点提供了与 Amazon Rekognition 的可靠、可扩展的连接，并且不需要互联网网关、网络地址转换 (NAT) 实例或 VPN 连接。有关更多信息，请参阅《Amazon VPC 用户指南》中的什么是 Amazon VPC。

接口 VPC 终端节点由 AWS 启用 PrivateLink。此 AWS 技术通过使用具有私有 IP 地址的弹性网络接口来实现 AWS 服务之间的私有通信。

> **Note**
>
> AWS 支持所有亚马逊 Rekognition 联邦信息处理标准 (FIPS) 终端节点。 PrivateLink

为 Amazon Rekognition 创建 Amazon VPC 端点

您可以创建两种类型的 Amazon VPC 端点以与 Amazon Rekognition 一起使用。

- VPC 端点可与 Amazon Rekognition 操作一起使用。对于大多数用户而言，这是最合适的 VPC 端点类型。
- VPC 端点可用于针对以下端点的 Amazon Rekognition 操作：符合联邦信息处理标准 (FIPS) 出版物 140-2 美国政府标准。

要开始将 Amazon Rekognition 与您的 VPC 结合使用，请使用 Amazon VPC 控制台为 Amazon Rekognition 创建一个接口 VPC 端点。有关说明，请参阅创建接口端点中的过程“使用控制台创建到 AWS 服务的接口端点”。请注意以下过程步骤：

- 步骤 3 – 对于服务类别，选择 AWS 服务。

- 步骤 4 – 对于服务名称，选择以下选项之一：
  - com.amazonaws.region.rekognition – 为 Amazon Rekognition 操作创建 VPC 端点。
  - com.amazonaws.region.rekognition-fips – 为针对以下端点的 Amazon Rekognition 操作创建 VPC 端点：符合联邦信息处理标准 (FIPS) 出版物 140-2 美国政府标准。

有关更多信息，请参阅《Amazon VPC 用户指南》中的入门。

为 Amazon Rekognition 创建 VPC 端点策略

您可以为 Amazon Rekognition 的 Amazon VPC 端点创建一个策略，在该策略中指定以下内容：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问权限。

以下示例策略允许通过 VPC 端点连接到 Amazon Rekognition 的用户调用 `DetectFaces` API 操作。该策略将阻止用户通过 VPC 端点执行其他 Amazon Rekognition API 操作。

用户仍可从 VPC 外部调用其他 Amazon Rekognition API 操作。有关如何拒绝对 VPC 外部的 Amazon Rekognition API 操作的访问信息，请参阅Amazon Rekognition 基于身份的策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rekognition:DetectFaces"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Principal": "*"
        }
    ]
}
```

修改 Amazon Rekognition 的 VPC 端点策略

1. 打开位于 https://console.aws.amazon.com/vpc/ 的 Amazon VPC 控制台。
2. 如果还没有为 Amazon Rekognition 创建端点，请选择创建端点。然后选择 com.amazonaws。*Region*.rekognition 然后选择创建端点。
3. 在导航窗格中，选择端点。
4. 选择 com.amazonaws。***Region***.rekognition 端点，然后选择屏幕下半部分的 “策略” 选项卡。
5. 选择编辑策略并对策略进行更改。

Amazon Rekognition 的合规性验证

作为多项合规计划的一部分，第三方审计师评估亚马逊 Rekognition 的安全与合规性。 AWS 其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

有关特定合规计划范围内的 AWS 服务列表，请参阅按合规计划划分的范围内的 AWS 服务。有关一般信息，请参阅 AWS 合规性计划。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅在 AWS Artifact 中下载报告。

您在使用 Amazon Rekognition 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在上部署以安全性和合规性为重点的基准环境的步骤。 AWS
- HIPAA 安全与合规架构白皮书 — 本白皮书描述了公司如何使用来 AWS 创建符合 HIPAA 标准的应用程序。
- AWS 合规资源 — 此工作簿和指南集可能适用于您所在的行业和所在地。
- AWS Config— 该 AWS 服务评估您的资源配置在多大程度上符合内部实践、行业指导方针和法规。
- AWS Security Hub— 此 AWS 服务可全面了解您的安全状态 AWS ，帮助您检查是否符合安全行业标准和最佳实践。

Amazon Rekognition 中的故障恢复能力

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用

区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础结构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon Rekognition 还提供多项功能来帮助支持您的数据弹性和备份需求。

Amazon Rekognition 中的配置和漏洞分析

配置和 IT 控制是 AWS 和您（我们的客户）之间的共同责任。有关更多信息，请参阅责任 AWS 共担模型。

防止跨服务混淆代理

在中 AWS，当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生跨服务模拟。尽管调用服务不应具有适当的权限，但仍可操纵以对另一个客户的资源进行操作，这会导致代理混淆。

为了防止这种情况，我们 AWS 提供了一些工具，帮助您保护所有服务的数据，这些服务委托人已被授予访问您账户中资源的权限。

我们建议在资源策略中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以限制 Amazon Rekognition 为其他服务提供的资源访问权限。

如果 `aws:SourceArn` 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个键来限制权限。如果同时使用键和包含账户 ID 的 `aws:SourceArn` 值，则 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的账户在同一策略语句中使用时，必须使用相同的账户 ID。

如果您只希望将一个资源与跨服务访问相关联，请使用 `aws:SourceArn`。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 `aws:SourceAccount`。

`aws:SourceArn` 的值必须是 Rekognition 使用的资源的 ARN，其指定格式如下：`arn:aws:rekognition:region:account:resource`。

`arn:User` ARN 的值应为将调用视频分析操作的用户（担任角色的用户）的 ARN。

解决代理混淆问题的推荐方法，是将 `aws:SourceArn` 全局条件上下文键与完整资源 ARN 结合使用。

如果您不了解完整的资源 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符（*）的 `aws:SourceArn` 键。例如，`arn:aws:rekognition:*:111122223333:*`。

为了防止出现代理混淆的问题，请执行以下步骤：

1. 在 IAM 控制台的导航窗格中，选择角色选项。控制台将显示您当前账户的角色。
2. 选择要修改的角色的名称。您修改的角色应具有AmazonRekognitionServiceRole权限策略。选择信任关系选项卡。
3. 选择编辑信任策略。
4. 在编辑信任策略页面上，将默认 JSON 策略替换为使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文密钥中的一个或两个的策略。请参阅下面的示例策略。
5. 选择更新策略。

以下示例信任策略演示如何使用 Amazon Rekognition 中的 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键来防范混淆代理问题。

如果您正在处理存储视频和流视频，则可以在您的 IAM 角色中使用如下所示的策略：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"rekognition.amazonaws.com",
            "AWS":"arn:User ARN"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "StringLike":{

 "aws:SourceArn":"arn:aws:rekognition:region:111122223333:streamprocessor/*"
            }
         }
      }
   ]
}
```

如果您只处理存储视频，则可以在您的 IAM 角色中使用如下所示的策略（请注意，您不必包含指定 `streamprocessor` 的 `StringLike` 参数）：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"rekognition.amazonaws.com",
            "AWS":"arn:User ARN"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            }
         }
      }
   ]
}
```

Amazon Rekognition 中的基础设施安全性

作为一项托管服务，Amazon Rekognition 受到 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问亚马逊 Rekognition。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

监控 Amazon Rekognition

监控是保持 Amazon Rekognition 和您的其他 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供以下监控工具来监控 Rekognition、在出现错误时进行报告并在适当的时候采取自动化措施：

- Amazon 会实时 CloudWatch监控您的 AWS 资源和您运行 AWS 的应用程序。您可以收集和跟踪指标，创建自定义的控制平面，以及设置警报以在指定的指标达到您指定的阈值时通知您或采取措施。例如，您可以 CloudWatch 跟踪您的 Amazon EC2 实例的 CPU 使用率或其他指标，并在需要时自动启动新实例。有关更多信息，请参阅 Amazon CloudWatch 用户指南。
- Amazon Lo CloudWatch gs 使您能够监控、存储和访问来自亚马逊 EC2 实例和其他来源的日志文件。 CloudTrail CloudWatch 日志可以监视日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南。
- Amazon EventBridge 可用于实现 AWS 服务自动化，并自动响应系统事件，例如应用程序可用性问题或资源更改。来自 AWS 服务的事件几乎实时 EventBridge 地传送到。您可以编写简单的规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。有关更多信息，请参阅 Amazon EventBridge 用户指南。
- AWS CloudTrail捕获由您的账户或代表您的 AWS 账户进行的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以识别哪些用户和帐户拨打了电话 AWS、发出呼叫的源 IP 地址以及呼叫发生的时间。有关更多信息，请参阅 用户指南。AWS CloudTrail

使用亚马逊监控 Rekognition CloudWatch

借助 CloudWatch，您可以获取单个 Rekognition 操作的指标或账户的全局 Rekognition 指标。您可以使用指标来跟踪基于 Rekognition 的解决方案的运行状况，并设置警报，以便在一个或多个指标超出定义的阈值时通知您。例如，您可以查看已发生的服务器错误数的指标或已检测到的人脸数的指标。您还可以查看特定 Rekognition 操作成功的次数的指标。要查看指标，您可以使用亚马逊 CloudWatch AWS Command Line Interface、亚马逊或 CloudWatch API。

您还可以使用 Rekognition 控制台查看选定时段内的聚合指标。有关更多信息，请参阅 练习 4：查看聚合指标（控制台）。

使用 Rekognition 的 CloudWatch 指标

要使用指标，您必须指定以下信息：

- 指标维度或无维度。维度 是帮助您对某指标进行唯一标识的名称/值对。Rekognition 具有一个名为操作的维度。它提供了特定操作的指标。如果您未指定维度，则指标的范围限定为账户内的所有 Rekognition 操作。
- 指标名称，如 `UserErrorCount`。

你可以使用、或 API 获取 Rekognition AWS Management Console的监控数据。 AWS CLI CloudWatch 您也可以通过其中一个 Amazon AWS 软件开发套件 (SDKs) 或 CloudWatch API 工具来使用 API。 CloudWatch 控制台根据来自 CloudWatch API 的原始数据显示一系列图表。根据您的需求差异，您可能倾向于使用控制台中显示的图表，也可能倾向于检索自 API 的图表。

下面的列表显示这些指标的一些常见用途。这些是入门建议，并不全面。

如何？	相关指标
如何跟踪识别的人脸数？	监控 `DetectedFaceCount` 指标的 Sum 统计数据。
如何获知我的应用程序是否已达到每秒最大请求数？	监控 `ThrottledCount` 指标的 Sum 统计数据。
如何监控请求错误？	使用 `UserErrorCount` 指标的 Sum 统计数据。
如何查找请求总数？	使用 `ResponseTime` 指标的 `ResponseTime` 和 `Data Samples` 统计数据。这包括任何导致错误的请求。如果您希望仅查看成功的操作调用，请使用 `SuccessfulRequestCount` 指标。
如何监控 `Rekognition` 操作调用的延迟？	使用 `ResponseTime` 指标。
如何监控 `IndexFaces` 成功将人脸添加到 Rekognition 集合的次数？	使用 `SuccessfulRequestCount` 指标和 `IndexFaces` 操作监控 Sum 统计数据。使用 `Operation` 维度选择操作和指标。

你必须拥有相应的 CloudWatch 权限才能监控 Rekognition。 CloudWatch有关更多信息，请参阅 Amazon CloudWatch 的身份验证和访问控制。

访问 Rekognition 指标

以下示例展示了如何使用 CloudWatch 控制台、 AWS CLI和 API 访问 Rekognition 指标。

CloudWatch

要查看指标（控制台）

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 选择 Metrics，选择 All Metrics 选项卡，然后选择 Rekognition。
3. 选择 Metrics with no dimensions，然后选择指标。

   例如，选择 DetectedFace 指标测量已检测到的人脸数。
4. 选择日期范围的值。图表中显示指标计数。

查看一段时间内发出的成功 **DetectFaces** 操作调用的指标 (CLI)。

- 打开 AWS CLI 并输入以下命令：

  ```
  aws cloudwatch get-metric-statistics --metric-name
  SuccessfulRequestCount --start-time 2017-1-1T19:46:20 --end-time
  2017-1-6T19:46:57 --period 3600 --namespace AWS/Rekognition --
  statistics Sum --dimensions Name=Operation,Value=DetectFaces --region
  us-west-2
  ```

  此示例显示一段时间内成功的 DetectFaces 操作调用。有关更多信息，请参阅 get-metric-statistics。

访问指标 (CloudWatch API)

- 调用 GetMetricStatistics。有关更多信息，请参阅 Amazon CloudWatch API 参考。

创建警报

您可以创建一个 CloudWatch 警报，在警报状态发生变化时发送亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 消息。警报会每隔一段时间（由您指定）监控一个指标，并根据相对于给定阈值的指标值每隔若干个时间段执行一项或多项操作。操作是一个发送到 Amazon SNS 主题或 Auto Scaling 策略的通知。

警报仅针对持续的状态变化调用操作。 CloudWatch 警报不会仅仅因为它们处于特定状态就调用操作。该状态必须改变并在指定数量的时间段内一直保持。

设置警报（控制台）

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 选择创建警报。创建警报向导随即启动。
3. 在 Metrics with no dimensions 指标列表中，选择 Rekognition Metrics，然后选择指标。

   例如，选择 DetectedFaceCount 以设置检测到的人脸的最大数目的警报。
4. 在 Time Range 区域中，选择一个日期范围值，其中包括已调用的人脸检测操作。选择下一步。
5. 填写名称和描述。对于每当，选择 >=，然后输入您选择的最大值。
6. 如果 CloudWatch 要在达到警报状态时向您发送电子邮件，请在 “每当此警报:” 中选择 “状态为警报”。要将警报发送到现有 Amazon SNS 主题，对于发送通知到：，请选择现有 SNS 主题。要为新的电子邮件订阅列表设置姓名和电子邮件地址，请选择 C reate topic 会 CloudWatch 保存列表并将其显示在字段中，这样您就可以用它来设置 future 警报。

   > **Note**
   >
   > 如果您使用创建主题创建一个新 Amazon SNS 主题，那么电子邮件地址在计划收件人接收通知之前必须通过验证。Amazon SNS 仅在警报进入警报状态时发送电子邮件。如果在验证电子邮件地址之前警报状态发生了变化，那么目标收件人不会接收到通知。
7. 预览警报预览部分中的警报。选择创建警报。

设置警报 (AWS CLI)

- 打开 AWS CLI 并输入以下命令。更改 `alarm-actions` 参数的值以引用您之前创建的 Amazon SNS 主题。

```
aws cloudwatch put-metric-alarm --alarm-name UserErrors --
alarm-description "Alarm when more than 10 user errors occur"
--metric-name UserErrorCount --namespace AWS/Rekognition --
statistic Average --period 300 --threshold 10 --comparison-
operator GreaterThanThreshold --evaluation-periods 2 --alarm-actions
arn:aws:sns:us-west-2:111111111111:UserError --unit Count
```

此示例演示如何为 5 分钟内出现 10 个以上的用户错误的情况创建警报。有关更多信息，请参阅 put-metric-alarm。

设置警报 (CloudWatch API)

- 调用 PutMetricAlarm。有关更多信息，请参阅 Amazon CloudWatch API 参考。

CloudWatchRekognition 的指标

本节包含有关适用于亚马逊 Rekognition 的亚马逊 CloudWatch 指标和操作维度的信息。

您也可以从 Rekognition 控制台查看 Rekognition 指标的聚合视图。有关更多信息，请参阅 练习 4：查看聚合指标（控制台）。

CloudWatch Rekognition 的指标

下表汇总了 Rekognition 指标。

指标	描述
SuccessfulRequestCount	成功的请求数。成功请求的响应代码范围为 200 至 299。 单位：计数 有效统计数据：Sum,Average
ThrottledCount	受限制请求的数目。当 Rekognition 接收的请求数超出为账户设置的每秒事务数限制时，它会限制请求。如果经常超出为账户设置的限制，您可以请求提高限制。要请求提高限制，请参阅 AWS 服务限制。 单位：计数 有效统计数据：Sum,Average
ResponseTime	Rekognition 用来计算响应的时间（以毫秒为单位）。 单位： 1. Data Samples 统计数据的数量

指标	描述
	2. Average 统计数据的毫秒数 有效统计数据：Data Samples,Average Note ResponseTime 指标未包含在 Rekognition 指标窗格中。
DetectedFaceCount	使用 IndexFaces 或 DetectFaces 操作检测到的人脸的数目。 单位：计数 有效统计数据：Sum,Average
DetectedLabelCount	使用 DetectLabels 操作检测到的标签的数目。 单位：计数 有效统计数据：Sum,Average
ServerErrorCount	服务器错误的数量。服务器错误的响应代码范围为 500 到 599。 单位：计数 有效统计数据：Sum, Average
UserErrorCount	用户错误（参数无效、图像无效、无权限等）的数目。用户错误的响应代码范围为 400 到 499。 单位：计数 有效统计数据：Sum,Average
MinInferenceUnits	StartProjectVersion 请求期间指定的最小推理单位数。 单位：计数 有效统计数据：Average

指标	描述
MaxInferenceUnits	`StartProjectVersion` 请求期间指定的最大推理单位数。 单位：计数 有效统计数据：`Average`
DesiredInferenceUnits	Rekognition 要增加或缩减到的推理单位数。 单位：计数 有效统计数据：`Average`
InServiceInferenceUnits	模型正在使用的推理单元数。 单位：计数 有效统计数据：`Average` 建议您使用平均值统计数据，获取实例使用数量的 1 分钟平均值。

CloudWatch Rekognition 直播的指标

Rekognition 还有第二个用于流式处理操作的命名空间，即“Rekognition 流式处理”。下表汇总了 Rekognition 流式处理指标。

指标	描述
SuccessfulRequestCount	成功的请求数。成功请求的响应代码范围为 200 至 299。 单位：计数 有效统计数据：`Sum,Average`
CallCount	在您的账户中执行的指定操作的数量。 有效统计数据：`Sum,Average`

指标	描述
ThrottledCount	受限制请求的数目。当 Rekognition 接收的请求数超出为账户设置的每秒事务数限制时，它会限制请求。如果经常超出为账户设置的限制，您可以请求提高限制。要请求提高限制，请参阅 AWS 服务限制。 单位：计数 有效统计数据：Sum,Average
ServerErrorCount	服务器错误的数量。服务器错误的响应代码范围为 500 到 599。 单位：计数 有效统计数据：Sum, Average
UserErrorCount	用户错误（参数无效、图像无效、无权限等）的数目。用户错误的响应代码范围为 400 到 499。 单位：计数 有效统计数据：Sum,Average
ConcurrentSessions	您的账户当前使用的并发会话数。 单位：计数 有效统计数据：Max,Average

CloudWatch Rekognition 的维度

要检索操作特定的指标，请使用 Rekognition 命名空间并提供操作维度。

有关尺寸的更多信息，请参阅 Amazon CloudWatch 用户指南中的尺寸。

CloudWatch Rekognition 自定义标签的尺寸

下表显示了可用于 Rekognition 自定义标签的 CloudWatch 尺寸：

维度	描述
ProjectName	您使用 `CreateProject` 创建的 Rekognition Custom Labels 项目的名称。
VersionName	您使用 `CreateProjectVersion` 创建的 Rekognition Custom Labels 项目版本的名称。

有关尺寸的更多信息，请参阅 Amazon CloudWatch 用户指南中的尺寸。

使用记录亚马逊 Rekognition API 调用 AWS CloudTrail

Amazon Rekognition 与一项服务 AWS CloudTrail集成，可记录用户、角色或服务在 Amazon Rekognition 中采取的操作。 AWS CloudTrail 将亚马逊 Rekognition 的所有 API 调用捕获为事件。捕获调用中包括通过 Amazon Rekognition 控制台的调用和对 Amazon Rekognition API 操作的代码调用。如果您创建跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括亚马逊 Rekognition 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。通过收集的信息 CloudTrail，您可以确定向 Amazon Rekognition 发出的请求、发出请求的 IP 地址、谁提出请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

亚马逊 Rekognition 中的信息 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当 Amazon Rekognition 中发生活动时，该活动会与其他服务事件 AWS 一起记录 CloudTrail 在事件历史记录中。您可以在自己的 AWS 账户中查看、搜索和下载最近发生的事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 AWS 账户中的事件，包括亚马逊 Rekognition 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail

- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有亚马逊 Rekognition 操作均由亚马逊 Rekogniti on API 参考 CloudTrail 记录并记录在案。例如，调用CreateStreamProcessor和DetectCustomLabels操作会在 CloudTrail 日志文件中生成条目。CreateCollection

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 Amazon Rekognition 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序出现。

以下示例显示了包含以下 API 操作的 CloudTrail 日志条目：StartLabelDetection和DetectLabels。

```
{
    "Records": [

        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
```

```
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T20:10:09Z"
                    }
                }
            },
            "eventTime": "2020-06-30T20:42:14Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "StartLabelDetection",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "video": {
                    "s3Object": {
                        "bucket": "my-bucket",
                        "name": "my-video.mp4"
                    }
                }
            },
            "responseElements": {
                "jobId":
"653de5a7ee03bd5083edde98ea8fce5794fcea66d077bdd4cfb39d71aff8fc25"
            },
            "requestID": "dfcef8fc-479c-4c25-bef0-d83a7f9a7240",
            "eventID": "b602e460-c134-4ecb-ae78-6d383720f29d",
            "readOnly": false,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
```

```
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T21:19:18Z"
                    }
                }
            },
            "eventTime": "2020-06-30T21:21:47Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DetectLabels",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "image": {
                    "s3Object": {
                        "bucket": "my-bucket",
                        "name": "my-image.jpg"
                    }
                }
            },
            "responseElements": null,
            "requestID": "5a683fb2-aec0-4af4-a7df-219018be2155",
            "eventID": "b356b0fd-ea01-436f-a9df-e1186b275bfa",
            "readOnly": true,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

Amazon Rekognition 中的准则和配额

以下几个部分提供了使用 Amazon Rekognition 时的准则和配额。有两种配额。无法更改设置的配额，例如最大图像大小。AWS 服务限额页面上列出的默认限额可通过 默认限额 部分中描述的步骤进行更改。

主题

- 支持的区域
- 设置配额
- 默认限额

支持的区域

有关可用 Amazon Rekognition 的 AWS 区域列表，请参阅《亚马逊网络服务通用参考》中的 AWS 区域和终端节点。

设置配额

以下是 Amazon Rekognition 中不可更改的限制列表。有关您可更改的限制的信息，例如每秒事务数 (TPS) 限制，请参阅 默认限额。

有关 Amazon Rekognition Custom Labels 限制，请参阅 Amazon Rekognition Custom Labels 中的准则和配额。

Amazon Rekognition Image

- 存储为 Amazon S3 对象的最大图像大小限制为 15 MB。
- `DetectModerationLabels` 最大图像尺寸的宽和高均为 10K 像素。
- `DetectLabels` 最大图像尺寸的宽和高均为 10K 像素。
- 要被检测到，人脸在 1920X1080 像素的图像中的大小不得小于 40x40 像素。对于尺寸大于 1920X1080 像素的图像，人脸的尺寸也需要相应增加。
- 最小图像尺寸的高和宽均为 80 像素。`DetectProtectiveEquipment` 最小图像尺寸的高和宽均为 64 像素。
- `DetectProtectiveEquipment` 最大图像尺寸的宽和高均为 4096 像素。

- 要被 DetectProtectiveEquipment 检测到，人脸在 800x1300 像素的图像中的大小不得小于 100x100 像素。对于尺寸大于 800x1300 像素的图像，最小人脸的尺寸也需要相应增加。
- 作为参数传入 API 的最大图像大小（原始字节数）为 5 MB。DetectProtectiveEquipment API 的限制为 4 MB。
- Amazon Rekognition 支持 PNG 和 JPEG 图像格式。也就是说，您作为输入提供给各种 API 操作（例如 DetectLabels 和 IndexFaces）的图像必须采用受支持的格式之一。
- 一个人脸集合中可存储的人脸向量最大数目为 2000 万。
- 一个人脸集合中可存储的人脸向量的默认最大数目为 1000 万。
- 搜索 API 返回的最大匹配人脸向量为 4096。
- 搜索 API 返回的最大匹配用户向量为 4096。
- DetectText 可以检测图像中最多 100 个单词。
- DetectProtectiveEquipment 最多可以检测到 15 个人身上的个人防护设备。

有关图像和人脸比较的最佳实践信息，请参阅传感器、输入图像和视频的最佳实践。

Amazon Rekognition Image 批量分析

- Amazon Rekognition Image 批量分析可以分析多达 1 万张图像。
- Amazon Rekognition Image 批量分析支持大小不超过 50MB 的输入清单。

Amazon Rekognition Video 存储视频

- Amazon Rekognition Video 可以分析存储的大小不超过 10 GB 的视频。
- Amazon Rekognition Video 可以分析长达 6 小时的存储视频。
- Amazon Rekognition Video 最多支持每个账户 20 个并发任务。
- 存储视频必须使用 H.264 编解码器进行编码。支持的文件格式为 MPEG-4 和 MOV。
- 任何分析音频数据的 Amazon Rekognition Video API 仅支持 AAC 音频编解码器。
- 分页标记的生存时间 (TTL) 期间为 24 小时。分页标记位于 Get 操作返回的 NextToken 字段中，例如 GetLabeldetection。

Amazon Rekognition Video 流视频

- 一个 Kinesis Video 输入流最多可关联一个 Amazon Rekognition Video 流处理器。

- 一个 Kinesis Data 输出流最多可关联一个 Amazon Rekognition Video 流处理器。
- 与一个 Amazon Rekognition Video 流处理器关联的 Kinesis Video 输入流和 Kinesis Data 输出流无法由多个处理器共享。
- 任何分析音频数据的 Amazon Rekognition Video API 仅支持 ACC 音频编解码器。

默认限额

默认配额列表可在 AWS 服务限额中找到。这些限制为默认值，可以更改。要请求提高限制，请创建案例。要查看您当前的配额限制（应用的配额值），请参阅 Amazon Rekognition 服务限额。要查看亚马逊 Rekognition Image 的 TPS 使用历史记录 APIs，请参阅亚马逊 Rekogniti on 服务配额页面，然后选择特定的 API 操作以查看该操作的历史记录。

主题

- 计算 TPS 限额更改
- TPS 限额的最佳实践
- 创建更改 TPS 配额的案例

计算 TPS 限额更改

您要求的新限额是多少？ 每秒事务数 (TPS) 在预期工作负载达到峰值时最为相关。了解工作负载峰值时的最大并发 API 调用量和响应时间（5-15 秒）非常重要。请注意，最少应为 5 秒。下面是两个示例：

- 示例 1：我预计在最繁忙时段开始时的最大并发面部认证 (CompareFaces API) 用户数为 1000。这些响应将在 10 秒钟内传播。因此，我所在的相关地区的 CompareFaces API 所需的 TPS 为 100 (1000/10)。
- 示例 2：在我最繁忙的时段开始时，预计的最大并发对象检测 (DetectLabels API) 调用数为 250。这些响应将在 5 秒钟内传播。因此，我所在的相关地区的 DetectLabels API 所需的 TPS 为 50 (250/5)。

TPS 限额的最佳实践

每秒事务数 (TPS) 的推荐最佳实践包括平滑峰值流量、配置重试次数以及配置指数回退和抖动。

1. 平稳的峰值流量。流量激增会影响吞吐量。要获得分配的每秒事务数 (TPS) 的最大吞吐量，请使用队列无服务器架构或其他机制来“平滑”流量，使其更加一致。有关使用 Rekognition 进行无服务器大

规模图像和视频处理的代码示例和参考，请参阅使用 Amazon Rekognition 进行大规模图像和视频处理。

2. 配置重试次数。请遵循 the section called “错误处理” 中的准则，为允许出现的错误配置重试次数。
3. 配置指数回退和抖动。在配置重试次数时配置指数回退和抖动可以提高可实现的吞吐量。请参阅中的错误重试和指数退避。 AWS

创建更改 TPS 配额的案例

要创建案例，请前往创建案例并回答以下问题：

- 您是否实施了the section called “TPS 限额的最佳实践”来平滑流量峰值并配置重试次数、指数回退和抖动？
- 您是否计算过所需的 TPS 配额变更？ 如果不是，请参阅the section called “计算 TPS 限额更改”。
- 您是否查看了 TPS 使用历史记录，以便更准确地预测未来的需求？ 要查看您的 TPS 使用历史记录，请参阅 Amazon Rekognition 服务限额页面。
- 您的使用案例是什么？
- APIs 你打算用什么？
- 你打算 APIs 在哪些地区使用这些？
- 您能否将负载分散到多个区域？
- 您每天处理多少张图片？
- 您预计这个交易量能维持多久（是一次性激增还是持续增长）？
- 您是如何被默认限制阻止的？ 查看以下异常表以确认您遇到的情况。

错误代码	例外	消息	这表示什么？	是否可以重试？
HTTP 状态代码 400	ProvisionedThroughputExceededException	超出了预配置的速率。	表示节流。您可以重试或评估提高限额的请求。	是
HTTP 状态代码 400	ThrottlingException	减慢；请求速率突然增加。	您可能会发送峰值流量并遇到节流。您应该调整流量，使其更加流畅和一致。然	是

错误代码	例外	消息	这表示什么？	是否可以重试？
			后配置其他重试次数。请参见最佳实践。	
HTTP 状态代码 5xx	ThrottlingException (HTTP 500)	服务不可用	表示后端正在纵向扩展以支持该操作。您应该重试该请求。	是

要详细了解错误代码，请参阅the section called “错误处理”。

> **Note**
>
> 这些限制取决于您所在的区域。请求更改限制会影响您请求的区域中的 API 操作。其他 API 操作和区域不受影响。

Amazon Rekognition 文档历史记录

下表描述《Amazon Rekognition 开发人员指南》每次发布时进行的重要修改。要获得本文档的更新通知，您可以订阅 RSS 源。

- 上次文档更新日期：2023 年 6 月 15 日

变更	说明	日期
Amazon Rekognition 现在支持新的审核标签，并提高了图像内容审核的准确性	Amazon Rekognition 的内容审核功能已得到增强，提高了准确性，可检测新标签，并可识别动画和/或插图内容。	2024 年 2 月 1 日
Amazon Rekognition 现在支持批量图像分析	Amazon Rekognition 现在支持通过在操作中使用清单文件异步处理大量图像。StartMediaAnalysisJob	2023 年 10 月 23 日
Amazon Rekognition 现在支持使用适配器进行自定义内容审核	现在，Amazon Rekognition 通过使用可扩展现有 Rekognition 深度学习模型功能 DetectModerationLabels 的适配器，支持提高 API 的准确性。	2023 年 10 月 12 日
Rekognition 现在支持带有集合的用户向量	Rekognition 人脸集合现在支持创建用户向量。用户向量汇总了同一用户的多个人脸向量，能更可靠地描绘用户，从而提高准确性。	2023 年 6 月 12 日
以下托管策略中添加了涉及管理用户的操作：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 在 `AmazonRekognitionReadOnlyAccess` 托管策略中添加了以下操作：`ListUsers` 、`SearchUse`	2023 年 6 月 12 日

	rs 、SearchUsersByImage	
Amazon Rekognition Image 现在可以推断视线方向	Amazon Rekognition Image 的人脸检测操作已得到改进，该操作现在可以推断出检测到的人脸的视线方向。	2023 年 5 月 31 日
Rekognition 内容审核 API 已改进	Rekognition 改进了图像和视频审核的内容审核模型。这一改进显著扩大了对露骨内容、暴力内容和暗示性内容的检测。现在，客户可以更准确地检测露骨和暴力内容，从而改善终端用户体验，保护其品牌身份，并确保所有内容都符合其行业法规和政策。	2023 年 5 月 9 日
Amazon Rekognition Image 现在可以检测被遮挡的人脸	Amazon Rekognition Image 现在可以检测被遮挡的人脸。Amazon Rekognition DetectFaces Imag IndexFaces APIs e's and 返回了一个新 FaceOccluded 属性，它表示图像中的脸部是部分捕捉到的，还是由于物体、衣服和身体部位重叠而无法完全看见。	2023 年 5 月 5 日
Rekognition 现在可以检测面部活跃度	Amazon Rekognition Video 现在可以用来检测视频中的活跃度，从而验证摄像机前的用户是否真实在场。Face Liveness 探测器还可以检测出现在摄像头前的仿冒攻击或绕过摄像头的攻击。	2023 年 4 月 11 日

Amazon Rekognition Video 更新。	Amazon Rekognition Video 现在可以检测更多标签并返回有关图像和标签属性的更多信息。 GetLabelDetection API 现在会返回有关别名和类别的信息。可以使用“纳入”和“排除”筛选选项筛选返回的标签信息。结果可以按时间戳或视频片段进行汇总。	2022 年 12 月 7 日
Amazon Rekognition Image 更新。	Amazon Rekognition Image 现在可以检测到更多标签，它现在可以返回有关图像和标签属性的更多信息。 DetectLabels API 现在会返回有关别名、类别和图像属性（如主色）的信息。可以使用“纳入”和“排除”筛选选项筛选返回的标签信息。	2022 年 11 月 11 日
针对 ProjectPolicy 和自定义标签模型复制的操作已添加到以下托管策略中：AmazonRekognitionReadOnlyAccess	Amazon Rekognition 在 `AmazonRekognitionReadOnlyAccess` 托管策略中添加了以下操作：`ListProjectPolicies`	2022 年 7 月 21 日
针对 ProjectPolicy 和自定义标签模型复制的操作已添加到以下托管策略中：AmazonRekognitionFullAccess，AmazonRekognitionCustomLabelsFullAccess	Rekognition 在 `AmazonRekognitionCustomLabelsFullAccess` 和 `AmazonRekognitionFullAccess` 托管策略中添加了以下操作：`CopyProjectVersion` 、`PutProjectPolicy` 、`ListProjectPolicies` 、`DeleteProjectPolicy`	2022 年 7 月 21 日

Amazon Rekognition Video 现在可以检测流视频中的标签	Amazon Rekognition Video 可以检测流视频中的宠物和包裹等标签。这是使用通过 `CreateStreamProcessor` 操作创建的流处理器上的 `ConnectedHome` 设置完成的。	2022 年 4 月 28 日
API 参考已从《Amazon Rekognition 开发人员指南》中移出	Amazon Rekognition API 参考现位于 Amazon Rekognition API 参考。	2022 年 2 月 24 日
以下托管策略的数据集管理更新：AWS 托管策略： AmazonRekognitionReadOnlyAccess、AWS 托管策略： AmazonRekognitionFullAccess、AWS 托管策略： AmazonRekognitionCustomLabelsFullAccess	Amazon Rekognition 在、和托管`CreateDataset` 策略`AmazonRekognitionFullOnlyAccess` 中`AmazonRekognitionReadOnlyAccess` 添加了以下操作`ListDatasetEntries` ：`AmazonRekognitionCustomLabelsFullAccess` `ListDatasetLabels` 、、、、、、`DescribeDataset` `UpdateDatasetEntries` `DistributeDatasetEntries` DeleteDataset	2021 年 11 月 1 日
目录中的一个新节点显示了托管在 Amazon Rekognition 上的示例 GitHub	为了便于访问，AWS 代码示例存储库中更新的代码示例现在显示在 Amazon Rekognition 开发者指南的单独节点中。	2021 年 10 月 22 日

Amazon Rekognition 可以检测视频片段中的黑色画面和主要节目内容	Amazon Rekognition 可以使用 `StartSegmentDetection` 和 `GetSegmentDetection` 操作将黑色画面、色条、片头字幕、片尾字幕、演播室徽标和主要节目内容识别为视频中的技术线索。	2021 年 6 月 7 日
以下托管策略的数据集管理更新：	您可以使用 Amazon Rekognition `DetectText` 操作检测图像中的多达 100 个单词。	2021 年 5 月 21 日
为AmazonRekognitionReadOnlyAccess和添加标签更新 AmazonRekognitionFullAccess	Rekognition 在 `AmazonRekognitionFullAccess` 和 `AmazonRekognitionReadOnlyAccess` 策略中添加了新的标记操作。	2021 年 4 月 2 日
Amazon Rekognition 现在支持标记	现在，您可以使用标签来识别、组织、搜索和筛选 Amazon Rekognition 集合、流处理器和自定义标签模型。	2021 年 3 月 25 日
Amazon Rekognition 现在可以检测个人防护设备	Amazon Rekognition 现在可以检测图像中人物的手套、口罩和头罩。	2020 年 10 月 15 日
Amazon Rekognition 有新的内容审核类别	Amazon Rekognition 内容审核类别现在包括 6 个新类别：毒品、烟草、酒精、赌博、粗鲁手势和仇恨符号。	2020 年 10 月 12 日
在本地显示来自 Kinesis Video Streams 的 Amazon Rekognition Video 结果的新教程	您可以在本地视频源中显示 Kinesis Video Streams 中流视频的 Amazon Rekognition Video 输出。	2020 年 7 月 20 日

关于使用 Gstreamer 的全新 Amazon Rekognition 教程	使用 Gstreamer，您可以通过 Kinesis Video Streams 将来自设备摄像头源的直播视频推送到 Amazon Rekognition Video。	2020 年 7 月 17 日
Amazon Rekognition 现在支持存储视频的分段	通过异步 Amazon Rekognition Video 分段 API，您可以检测存储视频中的黑色画面、色条、片尾字幕和镜头。	2020 年 6 月 22 日
Amazon Rekognition 现在支持 Amazon VPC 端点策略	通过指定策略，可以限制对 Amazon Rekognition Amazon VPC 端点的访问。	2020 年 3 月 3 日
Amazon Rekognition 现在支持检测存储视频中的文本	您可以使用 Amazon Rekognition Video API 异步检测存储视频中的文本。	2020 年 2 月 17 日
Amazon Rekognition 现在支持 Augmented AI（预览版）和 Amazon Rekognition Custom Labels	借助 Amazon Rekognition 自定义标签，您可以通过创建自己的机器学习模型来检测图像中的特殊对象、场景和概念。DetectModerationLabels 现在支持 Amazon Agumented AI（预览版）。	2019 年 12 月 3 日
亚马逊 Rekognition Amazon 现在支持 AWS PrivateLink	借助 AWS，您可以在 PrivateLink 您的 VPC 和 Amazon Rekognition 之间建立私有连接。	2019 年 9 月 12 日
Amazon Rekognition 人脸筛选	Amazon Rekognition 为 API 操作增加了增强的人脸过滤支持， IndexFaces 并为和 API 操作引入了人脸过滤功能 CompareFaces 。 SearchFacesByImage	2019 年 9 月 12 日

Amazon Rekognition Video 示例已更新	Amazon Rekognition Video 示例代码已更新，用于创建和配置 Amazon SNS 主题和 Amazon SQS 队列。	2019 年 9 月 5 日
添加了 Ruby 和 Node.js 示例	Amazon Rekognition Image 为同步标签和人脸检测添加了 Ruby 和 Node.js 示例。	2019 年 8 月 19 日
更新了不安全内容检测	Amazon Rekognition 不安全内容检测现在可以检测暴力内容。	2019 年 8 月 9 日
GetContentModeration 操作已更新	GetContentModeration 现在返回用于检测不安全内容的审核检测模型的版本。	2019 年 2 月 13 日
GetLabelDetection 并更新了 DetectModerationLabels 操作	GetLabelDetection 现在返回常见对象的边界框信息以及检测到的标签的分层分类。现在返回了用于标签检测的模型的版本。 DetectModerationLabels 现在返回用于检测不安全内容的模型的版本。	2019 年 1 月 17 日
DetectFaces 并且 IndexFaces 操作已更新	此版本更新了 DetectFaces 和 IndexFaces 操作。当 “属性” 输入参数设置为 “全部” 时，人脸位置地标包括 5 个新地标：upperJawlineLeft、 midJawlineLeft、ChinBottom、、midJawlineRight。 upperJawlineRight	2018 年 11 月 19 日
DetectLabels 操作已更新	现在为特定对象返回边界框。分层分类现在可用于标签。现在您可以获取用于检测的检测模型版本。	2018 年 11 月 1 日

IndexFaces 操作已更新	现在， IndexFaces 您可以使用 QualityFilter 输入参数过滤掉检测到的低质量人脸。您还可以使用 MaxFaces 输入参数根据人脸检测的质量和检测到的人脸的大小来减少返回的人脸数量。	2018 年 9 月 18 日
DescribeCollection 操作已添加	现在，您可以通过调用 DescribeCollection 操作来获取有关现有集合的信息。	2018 年 8 月 22 日
新的 Python 示例	在 Amazon Rekognition Video 内容中添加了 Python 示例，并对内容进行了重组。	2018 年 6 月 26 日
更新的内容布局	Amazon Rekognition Image 内容经过了重组，并添加了新的 Python 和 C# 示例。	2018 年 5 月 29 日
亚马逊 Rekognition 支持 AWS CloudTrail	Amazon Rekognition 与一项服务 AWS CloudTrail集成，可记录用户、角色或服务在 Amazon Rekognition 中采取的操作。 AWS 有关更多信息，请参阅使用 AW S 记录亚马逊 Rekognition API 调用。 CloudTrail	2018 年 4 月 6 日
分析存储视频和流视频。新的目录	有关分析存储视频的信息，请参阅使用存储视频。有关分析流视频的信息，请参阅使用流视频。已对 Amazon Rekognition 文档的目录进行重新排列以包含图像和视频操作。	2017 年 11 月 29 日

图像文本识别和人脸检测模型	Amazon Rekognition 现在可以检测图像中的文本。有关更多信息，请参阅检测文本。Amazon Rekognition 引入了对人脸检测深度学习模型的版本控制。有关更多信息，请参阅模型版本控制。	2017 年 11 月 21 日
名人识别	Amazon Rekognition 现在可以分析图像中是否有名人。有关更多信息，请参阅识别名人。	2017 年 6 月 8 日
图像监管	Amazon Rekognition 现在可以确定图像是否包含明显或暗示性的成人内容。有关更多信息，请参阅检测不安全的内容。	2017 年 4 月 19 日
检测到的人脸的年龄范围。汇总的 Rekognition 指标窗格	Amazon Rekognition 现在返回由 Rekognition API 检测到的人脸的估计年龄范围（以年为单位）。有关更多信息，请参阅 AgeRange。Rekognition 控制台现在有一个指标窗格，显示指定时间段内 Rekognition 的亚马逊 CloudWatch 指标汇总的活动图表。有关更多信息，请参阅练习 4：查看聚合指标（控制台）。	2017 年 2 月 9 日
新服务和指南	这是初始版本的图像分析服务、Amazon Rekognition 和《Amazon Rekognition 开发人员指南》。	2016 年 11 月 30 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。